# State-aware Video Procedural Captioning

Taichi Nishimura[1] Atsushi Hashimoto[2] Yoshitaka Ushiku[2] Hirotaka Kameko[3] Shinsuke Mori[3]
taichitary@gmail.com,{atsushi.hashimoto,yoshitaka.ushiku}@sinicx.com,{kameko,forest}@i.kyoto-u.ac.jp
[1]Graduate School of Informatics, Kyoto University, [2]OMRON SINIC X Corporation
[3]Academic Center for Computing and Media Studies, Kyoto University

## ABSTRACT

Video procedural captioning (VPC), which generates procedural text from instructional videos, is an essential task for scene understanding and real-world applications. The main challenge of VPC is to describe how to manipulate materials accurately. This paper focuses on this challenge by designing a new VPC task, generating a procedural text from the clip sequence of an instructional video and material list. In this task, the state of materials is sequentially changed by manipulations, yielding their state-aware visual representations (e.g., eggs are transformed into cracked, stirred, then fried forms). The essential difficulty is to convert such visual representations into textual representations; that is, a model should track the material states after manipulations to better associate the cross-modal relations. To achieve this, we propose a novel VPC method, which modifies an existing textual simulator for tracking material states as a visual simulator and incorporates it into a video captioning model. Our experimental results show the effectiveness of the proposed method, which outperforms state-of-the-art video captioning models. We further analyze the learned embedding of materials to demonstrate that the simulators capture their state transition. The code and dataset are available from https://github.com/misogil0116/svpc

## CCS CONCEPTS

• **Computing methodologies** → **Natural language generation**; **Scene understanding**; *Temporal reasoning*.

## KEYWORDS

procedural text, instructional video, simulator

**ACM Reference Format:**
Taichi Nishimura[1] Atsushi Hashimoto[2] Yoshitaka Ushiku[2] Hirotaka Kameko[3] Shinsuke Mori[3]. 2021. State-aware Video Procedural Captioning. In *Proceedings of the 29th ACM International Conference on Multimedia (MM '21), October 20–24, 2021, Virtual Event, China.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3474085.3475322

## 1 INTRODUCTION

In recent years, there has been significant progress in vision and language research that targets procedural text and instructional video [1, 2, 23, 24, 47]. Among the various tasks in such research, it is important to describe the content of the video using natural language for both scene understanding and real-world applications. For example, machines can help people learn new skills by providing a quick overview of instructional videos. To this end, video procedural captioning [37, 38] (VPC), which is the task of generating a procedural text from an instructional video, has been proposed. VPC requires a model (1) to segment important clips from a video, (2) to enumerate materials used in the clips, and (3) to describe how to manipulate them as a sentence for each clip. (1) and (2) have been independently studied as the task of temporal clip segmentation [12, 47] and object recognition [6, 33], respectively. In this paper, we focus on (3) by designing a new VPC task and method of generating a procedural text from a clip sequence pre-segmented in an instructional video and material list (Figure 1).

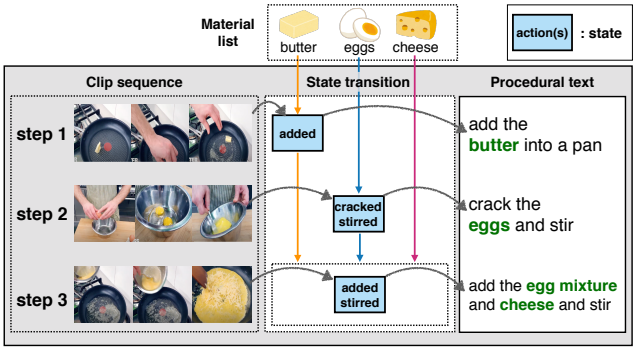



**Figure 1: Concept of the proposed method. Given a clip sequence and material list, the proposed method generates a procedural text by reasoning the state transition of materials at each step.**

Different from standard video captioning [11, 20, 40, 44, 46], our task requires a model to describe detailed material manipulations from visual observations. The manipulations change the state of materials sequentially, yielding their state-aware visual representations (e.g., in step 2 in Figure 1, "eggs" are transformed into "cracked" then "stirred"). The essential difficulty is to convert such visual representations into textual representations; that is, a model should track material states after manipulations to generate a procedural text. For example, when generating step 3 in Figure 1, the models should be aware that the yellow liquid in the bowl and white liquid in the pan correspond to the eggs and butter manipulated in steps 1 and 2, respectively. Existing video captioning models cannot track material states; thus, they miss materials, hallucinate them (e.g., say materials that are not in the clip), and lack details (e.g., say "cook ingredients"), thereby degrading the captioning performance. The above challenge is not specific to the cooking domain but is


*MM '21, October 20–24, 2021, Virtual Event, China*

ACM ISBN 978-1-4503-8651-7/21/10…$15.00
https://doi.org/10.1145/3474085.3475322

universal across multiple domains, such as wet-lab experiments and furniture assembly.

To address this challenge, this paper aims to generate procedural texts by modeling the state transition of materials from visual observations. The same motivation, tracking materials against their state transition, is shared by the research community in natural language understanding (NLU). To address this, some NLU studies have proposed a simulator to reason the state transition of materials in a procedural text [3, 5]. These simulators read one sentence in a procedural text, predict executed actions and involved materials, and recurrently update the state of materials to simulate their state transition.

Inspired by these ideas, we propose a novel VPC method, which modifies an existing NLU simulator as a *visual simulator* and incorporates it into an encoder-decoder architecture. Figure 1 illustrates the idea of the proposed method. Given a clip sequence and material list, the proposed method reasons the state transition of materials and generates a procedural text accurately. We emphasize that this work is the first attempt to integrate the NLU simulator into a video captioning model. In addition, based on the intuition that the state transition of materials is consistently traceable from the generated procedural texts, we attach a novel *textual re-simulator*, facilitating the model to generate a procedural text even more accurately.

In our experiments, we test the proposed method in the cooking domain because of its large variety of actions and materials (= ingredients). We compare it with two state-of-the-art video captioning models on four evaluations: word-overlap evaluation, ingredient prediction, retrieval evaluation, and qualitative analysis, with thorough ablation. Our experimental results show that the proposed method outperforms the state-of-the-art video captioning models. In addition, ablation studies show the effectiveness of integrating visual and textual simulators into the model. Finally, we analyze the learned embedding of materials to demonstrate that the simulators effectively capture the state transition of materials.

## 2 RELATED WORK

We describe the novelty of the proposed method in line with other works on video captioning in Section 2.1. In Section 2.2, we discuss simulators for procedural text understanding because they are the main components of the proposed method.

### 2.1 Video captioning

Video captioning is an attractive field for both computer vision and natural language processing communities. The task settings of video captioning vary according to the nature of the input video (e.g., one short clip, clip sequence, or long untrimmed video). Our VPC task targets a clip sequence and thus belongs to the video paragraph description [20, 30, 44], which aims to generate multiple sentences for a given clip sequence pre-segmented in the video.

In the video paragraph description, recently, Transformer-based [42] models have been featured because of their ability to capture long-term information in a clip sequence, compared with LSTM-based models [30, 44]. MART [20] is a transformer-based models that achieves state-of-the-art performance in the video paragraph description. The key contribution of MART is the introduction of a gated recurrent memory module, which makes it easy for the model to summarize important information in the previous step.

Different from standard video captioning, VPC requires a model to generate detailed material manipulations for input clips. To this end, previous VPC works [37, 38] target narrated videos and utilize transcription as a cue to generate a procedural text by fusing video and transcription features. Although these approaches work well for narrated videos, transcription is not always available for all instructional videos. Speaking during manipulations or adding narration to videos requires significant effort. Thus, the proposed VPC task extends the previous VPC works to describe manipulations even from non-narrated videos; our task allows a model to refer to a human-created material list, which will be replaced with a machine-recognized one in the future. Our task requires a model to track material states that are changed by manipulations to generate a procedural text accurately. To this end, we propose a novel VPC method, which modifies an NLU simulator as a visual simulator and incorporates it into a transformer-based encoder-decoder.

### 2.2 Procedural text understanding

In NLU, procedural texts are a popular target for understanding, and recipes have been featured in this field. Many researchers have proposed methods to understand recipes, which are roughly divided into two categories: (1) a graph-based parser and (2) a reasoning-based simulator.

A graph-based parser aims to train a model to map recipes into graph or tree structures. Kiddon *et al.* [18] and Maeta *et al.* [22] proposed a model to estimate a graph structure called the action graph and recipe flow graph, respectively. Jermsurawong and Nizar [17] proposed a parser with the SIMMR dataset, which provides a tree structure for understanding ingredient merging operations. Then, Pan *et al.* [28] and Nishimura *et al.* [27] provided new multimodal tree structure datasets and parsers, which can be regarded as an extension of the text-only version to a vision-and-language version. These parsers focus on enabling machines to interpret recipes as structural representations, rather than capturing the state transition of ingredients in recipes.

A reasoning-based simulator aims to model the state transition of ingredients in recipes by updating the ingredient states at each step. Gupta and Durrett [13] proposed a structured simulator that imposes constraints on the state transition of ingredients (e.g., stir-fried potatoes are not cut in later steps). Amac *et al.* [3] proposed a method to answer multi-modal question answering (QA) tasks by reasoning the state transition of ingredients using a relation reasoning network [35]. This paper builds upon the neural process network (NPN) [5], which learns to estimate the state transition of ingredients by predicting executed actions and involved ingredients. Owing to the ability to capture the state transition of ingredients, these simulators reported competitive results in QA tasks [3, 9]. Our work adopts a reasoning-based simulator. We modify an NPN as the visual simulator and incorporate it into the transformer-based encoder-decoder architecture to enable the model to capture the state transition of ingredients.

## 3 PROPOSED METHOD

In this section, we present the proposed method. After describing an overview in Section 3.1, we explain the components of the proposed

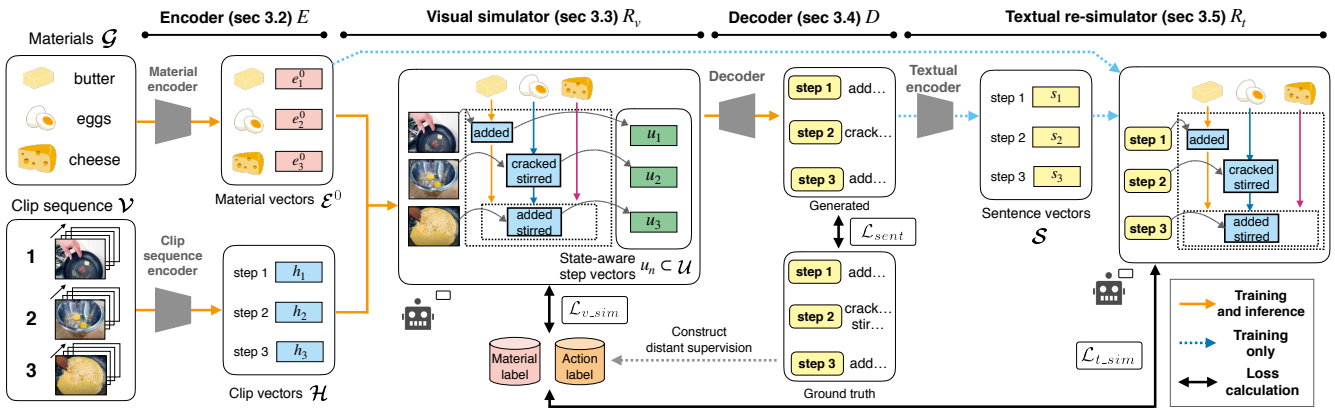



**Figure 2: An overview of the proposed method. To track material states in a clip sequence, we incorporate the visual simulator $R_v$ into the transformer-based encoder-decoder architecture ($E$ and $D$). In addition, based on our intuition that the state transition of materials is traceable from the generated procedural texts, we attach the textual re-simulator $R_t$ to the model.**

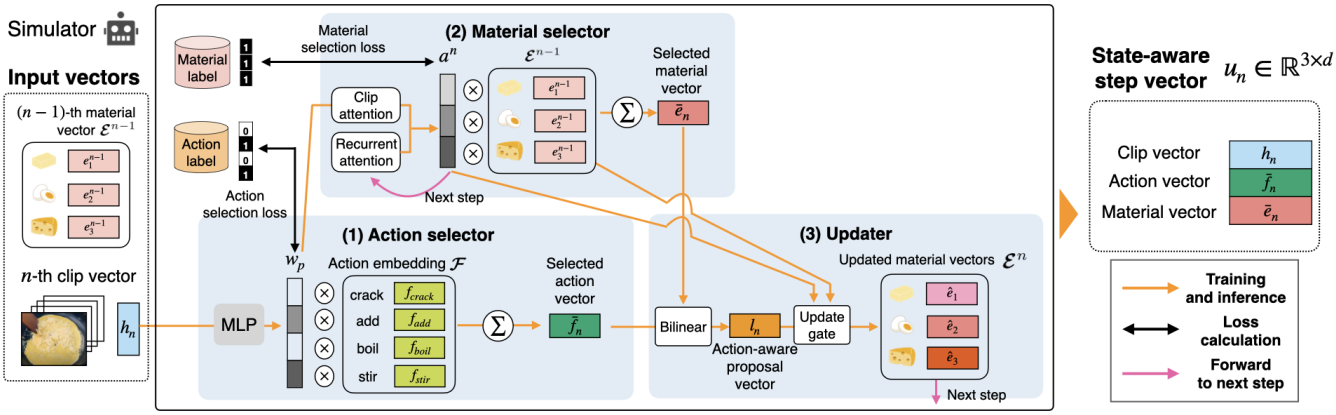



**Figure 3: An overview of the (visual) simulator. The simulator recurrently reasons the state transition of the materials at each step. Specifically, it predicts executed actions and involved materials in (1) the action and (2) material selector and then updates the state of materials in (3) the updater. The updated materials are forwarded to the next step. The textual re-simulator has the same modules.**

method from Section 3.2 to Section 3.5. Finally, in Section 3.6 we explain the loss functions to train the model.

## 3.1 Overview

Figure 2 shows an overview of the proposed method. Given a clip sequence $\mathcal{V} = (v_1, \ldots, v_n, \ldots, v_N)$ and a material list $\mathcal{G} = (g_1, \ldots, g_m, \ldots, g_M)$, our goal is to output a procedural text $\mathcal{Y} = (y_1, \ldots, y_n, \ldots, y_N)$ by recurrently generating sentences from corresponding clips. To generate a procedural text accurately, it is essential for models to track material states in the clip sequence (e.g., eggs are transformed into cracked, stirred, then fried forms).

Inspired by recent advances in NLU, we achieve this by modifying the existing reasoning-based NLU simulator NPN as the visual simulator $R_v$, and incorporate it into the transformer-based encoder-decoder architecture ($E$ and $D$). Specifically, given clip $\mathcal{H}$ and material encoded vectors $\mathcal{E}^0$, the visual simulator reasons the state transition of materials and outputs state-aware step vectors as $\mathcal{U} = R_v(\mathcal{H}, \mathcal{E}^0)$. Then, the decoder $D$ outputs a procedural text conditioned on $\mathcal{U}$. We observe that this simple integration of the visual simulator is effective for the model to generate a procedural text accurately.

In addition, based on our assumption that the state transition of materials should be consistently traceable from the generated procedural text, we attach a novel textual re-simulator $R_t$, encouraging the model to generate a procedural text even more accurately. Specifically, the textual re-simulator $R_t$ reasons the state transition of materials from the generated procedural text as $R_t(\mathcal{S}, \mathcal{E}^0)$, where $\mathcal{S}$ represents the encoded sentence vectors. Note that the textual re-simulator is only used during the training phase and detached during the inference phase.

## 3.2 Encoder

The input has two components: a material list $\mathcal{G}$ and a clip sequence $\mathcal{V}$. Thus, we develop a suitable encoder for each component.

**Material encoder.** To encode a material list, we input them to concatenated neural networks of pre-trained GloVe [31][1] word embedding and multi-layer perceptrons (MLPs) with the ReLU activation function. Multi-word materials (e.g., parmesan cheese) are represented by the average embedding vector of words. We then add positional encoding (PE) to material indices, and obtain the initial material vectors $\mathcal{E}^0 = (\boldsymbol{e}^0_1, \ldots, \boldsymbol{e}^0_m, \ldots, \boldsymbol{e}^0_M)$.

**Clip sequence encoder.** A clip sequence $\mathcal{V}$ is hierarchical because the clip sequence contains multiple clips, and each clip is composed of sequential frames. Thus, to encode a clip sequence effectively, we design a two-stage transformer suitable to encode a sequence of sequences. First, the former transformer encodes each clip into a feature vector by extracting the vector, which corresponds to the [CLS] token as in [10, 20, 39]. Then the latter transformer is trained over the sequence to obtain the step-aware clip vectors $\mathcal{H} = (\boldsymbol{h}_1, \ldots, \boldsymbol{h}_n, \ldots, \boldsymbol{h}_N)$ in the clip sequence.

## 3.3 Visual simulator

Based on the encoded vectors of the clip sequence and material list $(\mathcal{H}, \mathcal{E}^0)$, the visual simulator, shown in Figure 3, reasons the state transition of materials at each step. Specifically, at the $n$-th step, given the $n$-th clip $\boldsymbol{h}_n$ and $(n-1)$-th material list $\mathcal{E}^{n-1}$, the visual simulator predicts executed actions and involved materials in (1) the action and (2) material selector and then updates the state of materials in (3) the updater. After $n$-th reasoning, it outputs a state-aware step vector $\boldsymbol{u}_n \in \mathbb{R}^{3\times d}$, which concatenates the $n$-th clip $\boldsymbol{h}_n$, selected action $\bar{\boldsymbol{f}}_n$ and material vectors $\bar{\boldsymbol{e}}_n$ ($d$ represents the dimension of these vectors). The visual simulator recurrently repeats the above process until processing the end element of the clip sequence. This simulation process is the same as NPN except for the input modality; we replace the textual sentence and entity vectors for NLU with visual clip $\mathcal{H}$ and material vectors $\mathcal{E}^0$ for our task. Thus, we only provide a high-level overview in this subsection, and further details of the simulator are provided in Appendix A.

**(1) Action selector.** Given a clip vector $\boldsymbol{h}_n$, the action selector outputs the selected action vector $\bar{\boldsymbol{f}}_n$ by choosing actions executed in the clip from the predefined action embedding $\mathcal{F}$. For example, in Figure 3, the actions "crack" and "stir" are executed in the clip, thus both $\boldsymbol{f}_{crack}$ and $\boldsymbol{f}_{stir}$ should be selected. To consider multiple actions, the action selector computes a soft selection $\boldsymbol{w}_p$ as an action probability for each action in $\mathcal{F}$. Then it outputs the selected action vector $\bar{\boldsymbol{f}}_n$ as a weighted sum of the action embedding $\mathcal{F}$ and action probability $\boldsymbol{w}_p$.

**(2) Material selector.** Based on the action probability $\boldsymbol{w}_p$ and clip vector $\boldsymbol{h}_n$, the material selector outputs a selected material vector $\bar{\boldsymbol{e}}_n$ by choosing materials that are involved in the clip from the material list $\mathcal{E}^{n-1}$. For example, in Figure 3, the raw "cheese" and manipulated "eggs" and "butter" should be selected. To consider such a combination of raw and manipulated material selection, the material selector has two attention modules: (1) clip attention and (2) recurrent attention. While the clip attention selects materials from the current clip vector $\boldsymbol{h}_n$, the recurrent attention selects materials based on both the current and previous clips. Using these modules, the material selector computes a soft selection $\boldsymbol{a}^n$ as the material probability for each material in the material list $\mathcal{E}^{n-1}$. Then it outputs the selected material vector $\bar{\boldsymbol{e}}_n$ as a weighted sum of the material vectors $\mathcal{E}^{n-1}$ and material probability $\boldsymbol{a}^n$.

**(3) Updater.** Based on the selected actions and materials, the updater represents the state transition of materials by computing a new material vector $\hat{\boldsymbol{e}}_m$. To this end, it first calculates an action-aware proposal vector $\boldsymbol{l}_n$ of materials with a bilinear transformation of selected action and material vectors $(\bar{\boldsymbol{f}}_n, \bar{\boldsymbol{e}}_n)$. Then, based on the material probability $\boldsymbol{a}^n$, it computes the new material vector $\hat{\boldsymbol{e}}_m$ by interpolating the action-aware proposal vector $\boldsymbol{l}_n$ and the current material vector $\boldsymbol{e}^{n-1}_m$. The $m$-th new material vector $\hat{\boldsymbol{e}}_m$ is assigned to $\mathcal{E}^n_m$, which is forwarded to the next $(n+1)$-th step.

## 3.4 Decoder

The decoder $D$ outputs a procedural text by recurrently generating sentences from the $n$-th output vector $\boldsymbol{u}_n$ of the visual simulator. As our decoder $D$, we use the transformer, which achieves state-of-the-art performance in video captioning tasks [37, 38, 48]. Our task allows the model to refer to a material list, thus to encourage the decoder to generate materials, we incorporate the copy mechanism [36] into the decoder $D$. When generating the $k$-th word in the $n$-th sentence, given the updated materials $\boldsymbol{e}^n_m \in \mathcal{E}^n$, the copy mechanism first calculates the attention probability $\boldsymbol{\beta}^m_{n,k}$ using the bilinear dot product of the vectors of the decoder output $\boldsymbol{o}_{n,k}$ and each material $\boldsymbol{e}^n_m \in \mathcal{E}^n$ as:

$$\boldsymbol{\beta}^m_{n,k} = \frac{\exp\{(\boldsymbol{o}_{n,k})^\top \boldsymbol{W}_c \boldsymbol{e}^n_m\}}{\sum_i \exp\{(\boldsymbol{o}_{n,k})^\top \boldsymbol{W}_c \boldsymbol{e}^n_i\}}, \tag{1}$$

where $\boldsymbol{W}_c$ represents a bilinear map.

Then it calculates the copying gate $g_{n,k}$ ($0 \le g_{n,k} \le 1$), which makes a soft choice between selecting a material from the material list and generating a word from the vocabulary:

$$g_{n,k} = \sigma(\boldsymbol{W}_g[\boldsymbol{o}_{n,k}; \sum_m \boldsymbol{\beta}^m_{n,k} \boldsymbol{e}^n_m] + \boldsymbol{b}_g), \tag{2}$$

where $[\cdot]$, $\sigma(\cdot)$, $\boldsymbol{W}_g$, and $\boldsymbol{b}_g$ represent the concatenation function, sigmoid function, linear map, and bias, respectively. Based on $g_{n,k}$, the final predicted word probability $P_{n,k}(w)$ is computed as the weighted sum of the copy probability and generation probability as follows:

$$P_{n,k}(w) = \left(1 - g_{n,k}\right) P^{\text{voc}}_{n,k}(w) + g_{n,k} \left( \frac{1}{|\boldsymbol{g}_m|} \sum_{i: w_i \in \boldsymbol{g}_m} \boldsymbol{\beta}^i_{n,k} \right), \tag{3}$$

where $P^{\text{voc}}_{n,k}(w)$ and $|\boldsymbol{g}_m|$ represent the probability of the $n$-th sentence's $k$-th word $w$ in the vocabulary and the number of words of in the $m$-th material, respectively[2].

## 3.5 Textual re-simulator

Because the state transition of materials is identical in the visual and textual worlds, it should be consistently traceable from the generated procedural texts. Based on this assumption, we add a novel textual re-simulator $R_t$, encouraging the model to generate a procedural text even more accurately.

[1] We employ pre-trained 300D word embedding, which can be downloaded from http://nlp.stanford.edu/data/glove.6B.zip

[2] To consider multiple words of materials, we divide the probability by the number of words.

The textual re-simulator consists of two sub-modules: (1) a textual encoder and (2) a textual simulator. The textual encoder converts a generated procedural text into step-aware sentence vectors $\boldsymbol{S} = (\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n, \ldots, \boldsymbol{s}_N)$. First, it applies the straight-through version of Gumbel softmax resampling [16] to sample a procedural text, preserving the differentiable chain. The sampled procedural text is further converted into feature vectors by computing the average vector of the word embedding at each step. Note that the word embedding is shared between the decoder and textual encoder. They are then converted into step-aware sentence vectors $\boldsymbol{S}$ using a biLSTM encoder. Finally, based on $\boldsymbol{S}$, the textual simulator, another NPN described in Section 3.3, reasons the state transition of materials again as $R_t(\boldsymbol{S}, \mathcal{E}^0)$.

## 3.6 Loss functions

To train the model, we compute three types of losses: (1) sentence generation loss $\mathcal{L}_{sent}$, (2) visual simulation loss $\mathcal{L}_{v\_sim}$, and (3) textual re-simulation loss $\mathcal{L}_{t\_sim}$.

**(1) Sentence generation loss $\mathcal{L}_{sent}$.** This loss aims to train the decoder, and is computed as the summed negative log-likelihood for all input/output pairs $\{(\mathcal{V}, \mathcal{G}), \mathcal{Y}\}$ in the training set.

**(2) Visual simulation loss $\mathcal{L}_{v\_sim}$.** This loss aims to train the visual simulator, and consists of two losses: (1) material selection loss and (2) action selection loss. These losses are computed as the summed binary cross-entropy loss based on whether the materials/actions are involved/executed in the clip. To avoid costly human annotations, we compute the loss from distant supervision [26] following the original NPN training method [5]. For the material selection loss, labels are obtained whether or not each step contains materials in the material list; for the action selection loss, labels are obtained whether or not each step contains actions in the 384 actions defined by [5]. For example, from the sentence "crack the eggs and stir," "eggs" is extracted as a material label, and "crack" and "stir" are extracted as an action label. As the action selection loss, the simple binary cross-entropy does not work in our preliminary experiment because the ratio of positive to negative actions is imbalanced; a few actions are positive and most of the actions are negative. Thus, as the action selection loss, we use asymmetric loss [45], which is a weighted binary cross-entropy loss, to defuse the imbalanced problem.

**(3) Textual re-simulation loss $\mathcal{L}_{t\_sim}$.** To train the textual re-simulator, we also compute the above visual simulation loss from the sampled procedural texts.

**Total loss.** Consequently, to train the entire model in an end-to-end manner, the total loss is computed as

$$\mathcal{L}_{total} = \mathcal{L}_{sent} + \mathcal{L}_{v\_sim} + \lambda \mathcal{L}_{t\_sim}, \tag{4}$$

where $\lambda$ is a hyper-parameter used for weighting the importance of the textual re-simulation loss.

# 4 EXPERIMENTS

We test the proposed method in the cooking domain because procedural texts (= recipes) have a large variety of actions and materials (= ingredients). We compare it with two state-of-the-art video captioning models on four evaluations: word-overlap evaluation (Section 4.2), ingredient prediction (Section 4.3), retrieval evaluation (Section 4.4), and qualitative analysis (Section 4.5) with thorough ablation. We also visualize the learned embedding of ingredients, demonstrating that the visual simulator effectively reasons the state transition of ingredients (Section 4.6).

**Table 1: YouCook2-ingredient dataset statistics.**

| | train | val | test |
|---|---|---|---|
| #recipes | 1,233 | 100 | 329 |
| #steps / #recipes | 7.7 | 7.9 | 7.6 |
| #ingredients / #recipes | 10.4 | 10.2 | 10.3 |

## 4.1 Experimental settings

**Dataset.** We use the YouCook2 dataset [47], which consists of 2,000 cooking videos from 89 recipe categories. All of the videos have 3–16 clips with a start/end timestamp annotated by humans, and each clip is also annotated with an English sentence. Note that ingredients are not annotated in the original dataset; thus, we prepared the YouCook2-ingredient dataset by annotating ingredients with recipes. We downloaded the videos from the website, removed invalid videos (e.g., corrupted or unavailable videos), and finally obtained 1,664 videos. Then, we hired three annotators to write ingredients that appear in the recipe (for annotation details, see Appendix B). After this annotation process, we manually removed recipe/video pairs that had no ingredient annotations (here, two videos were removed). The remaining 1,662 valid videos were used as the YouCook2-ingredient dataset and divided as follows: 1,233 for training, 100 for validation, and 329 for testing[3]. Table 1 shows the statistics of the YouCook2-ingredient dataset.

**Data preprocessing.** As clip features, we use concatenated features of appearance and optical flow provided by [47]. With respect to the appearance, 2,048D feature vectors extracted from the "Flatten-673" layer in ResNet-200 [14] are used, and for the optical flow, 1,024D feature vectors extracted from the "global pool" layer in BN-Inception [15] are used. As in [20], we truncated sequences longer than 100 for the clip and 20 for the sentence and set the maximum length of the clip sequence to 12. Finally, we built the vocabulary based on words that occurred three times or more, and the resulting vocabulary contained 951 words.

**Hyper-parameter settings.** For both the encoder and decoder transformers, we set the hidden size to 768, the number of layers to two, and the number of attention heads to 12. We train the model following the optimization method described in [10, 20]; we use the Adam optimizer [19] with an initial learning rate of 0.0001, $\beta_1 = 0.9$, and $\beta_2 = 0.999$. The L2 weight decay is set to 0.01, and the learning rate warmup is over the first five epochs. We set the batch size to 16, and continue training at most 50 epochs using early stopping with CIDEr-D. We tune $\lambda$ with four different values $\lambda \in \{0.25, 0.5, 0.75, 1.0\}$ and set $\lambda$ to 0.5 in our experiments (for details, see Section 4.2).

**Models.** We test the proposed method by comparing it with two state-of-the-art video captioning models, as described below.

- **Transformer-XL** [8] is a powerful transformer-based language model that was originally proposed for capturing long-term dependency in natural language. As in [20], we adapt

[3]We will release annotated ingredients and the dataset split.

**Table 2: Word-overlap metrics for the baseline and the proposed models with ablation studies. The scores in bold are the best among the comparative models. "I" indicates whether the model uses ingredient information or not. B=BLEU, M=METEOR, C=CIDEr-D, RL=ROUGE-L.**

| Baseline | I | B1 | B4 | M | C | RL |
|---|---|---|---|---|---|---|
| Transformer-XL | | 37.8 | 6.7 | 14.8 | 25.0 | 31.5 |
| + Ingredients (Transformer-XL-I) | ✓ | 39.7 | 8.8 | 16.2 | 38.5 | 34.4 |
| MART | | 39.8 | 7.7 | 15.6 | 35.9 | 32.2 |
| + Ingredients (MART-I) | ✓ | 44.6 | 10.2 | 18.3 | 49.7 | 36.4 |
| Ours | | | | | | |
| Video only (V) | | 39.4 | 8.0 | 15.3 | 32.7 | 32.1 |
| V + Ingredients (VI) | ✓ | 45.3 | 11.2 | 19.3 | 61.7 | 37.4 |
| VI + Visual simulator (VIV) | ✓ | 48.3 | 11.9 | 21.2 | 73.8 | **40.0** |
| VIV + Textual re-simulator (VIVT) | ✓ | **49.4** | **12.1** | **21.6** | **77.3** | 39.9 |

**Table 3: Change in word-overlap metrics with controlled $\lambda$.**

| | B1 | B4 | M | C | RL |
|---|---|---|---|---|---|
| $\lambda = 0$ (VIV) | 48.3 | 11.9 | 21.2 | 73.8 | 40.0 |
| $\lambda = 0.25$ | 48.8 | 11.9 | 21.4 | 75.4 | 40.0 |
| $\lambda = 0.5$ | 49.4 | **12.1** | **21.6** | **77.3** | 39.9 |
| $\lambda = 0.75$ | **50.1** | 11.7 | **21.6** | 76.8 | 39.6 |
| $\lambda = 1.0$ | 49.2 | 12.0 | **21.6** | 76.1 | **40.1** |

it for our task; the model directly uses all of the previous hidden states to generate a current sentence.

- **MART** [20] is a transformer-based video captioning model that achieves state-of-the-art performance on the video paragraph description. This model generates a sentence with a gated recurrent memory module, which does not pass all of the previous hidden states, but effectively summarizes important information in the previous step.

Note that these models originally have no ingredient list in the inputs and copy mechanism in the decoder. Thus for a fair comparison, we prepare for additional baselines, baseline + ingredient (-I) models, which incorporate the material encoder (Section 3.2) and the copy mechanism into the models (for implementation details of the models, see Appendix C).

**Ablations.** To reveal the impact of the components in the proposed method, we conduct ablation studies on the following variations.

- **Video only (V)** encodes a clip sequence with the clip sequence encoder, then generates a procedural text.
- **V + Ingredient (VI)** incorporates the material encoder and copy mechanism in the model.
- **VI + Visual simulator (VIV)** incorporates the visual simulator into the VI model.
- **VIV + Textual re-simulator (VIVT)** additionally incorporates the textual re-simulator into the VIV model.

## 4.2 Word-overlap evaluation

**Scores.** To evaluate the captioning performance of the proposed method, we compute commonly used word-overlap metrics, such as BLEU [29], ROUGE-L [21], METEOR [4], and CIDEr-D [43] in the test set. Note that they are computed at the recipe (paragraph)-level following [20, 30, 44], not at the step (sentence)-level.

**Table 4: Results of ingredient prediction.**

| Baseline | Recall | Precision | F1 |
|---|---|---|---|
| Transformer-XL | 11.7 | 19.5 | 14.6 |
| + Ingredients (Transformer-XL-I) | 19.7 | 33.6 | 24.8 |
| MART | 13.0 | 20.7 | 16.0 |
| + Ingredients (MART-I) | 21.6 | 33.1 | 26.2 |
| Ours | | | |
| Video only (V) | 10.9 | 19.3 | 13.9 |
| V + Ingredients (VI) | 24.8 | 40.3 | 30.7 |
| VI + Visual simulator (VIV) | 34.7 | 50.0 | 40.8 |
| VIV + Textual re-simulator (VIVT) | **35.5** | **51.5** | **42.0** |

**Results.** Table 2 shows the results of the word-overlap evaluation. We observe that the proposed method consistently outperforms the state-of-the-art captioning models by a significant margin. Our ablation studies show that the VIV model performs better than the VI model, and the VIVT model further improves the VIV model. This indicates that both the visual simulator and the textual re-simulator are effective for generating a recipe accurately.

**Performance change of controlling the hyper-parameter $\lambda$.** Table 3 shows the results by varying the $\lambda \in \{0.0, 0.25, 0.5, 0.75, 1.0\}$. Note that $\lambda = 0$ is equivalent to the VIV model, which does not have a textual re-simulator. The results indicate that (1) the VIVT model performs better than the VIV model for any $\lambda$ values, and (2) $\lambda = 0.5$ performs the best among the three metrics (BLEU4, METEOR, and CIDEr-D) and obtains competitive results in BLEU1 and ROUGE-L. Thus, we set $\lambda = 0.5$ in our experiments.

## 4.3 Ingredient prediction

To evaluate whether the models use correct ingredients at each step without missing and hallucinating them, we design the ingredient prediction, which measures the step-level overlap of ingredients between generated and ground-truth recipes. To this end, we first construct an ingredient dictionary from all unique ingredients in the YouCook2-ingredient dataset. Then, at each step, we extract ingredients that are exact-matched between generated recipes and the ingredient dictionary. The same process is performed to extract ingredients in the ground-truth recipes. Finally, based on the extracted ingredient sets, we compute the micro- recall, precision, and F1 scores, respectively.

**Results.** Table 4 shows the results of the ingredient prediction. This result shows that the proposed method outperforms the state-of-the-art video captioning models. In our ablation, we observe the same tendency of performance change to the word-overlap evaluation. We note that the VIV model performs much better than the VI model by 10% in F1, indicating that not only the copy mechanism but also the visual simulator are important for generating ingredients correctly. We also notice that the VIVT model improves the VIV model by 1.2% in F1, demonstrating the effectiveness of the textual re-simulator.

## 4.4 Retrieval evaluation

To evaluate whether the generated procedural texts are sufficiently concrete to describe the input clips, we design a step-level zero-shot sentence-to-clip retrieval evaluation. As a retrieval model, we employ the MIL-NCE model [23] pre-trained on the HowTo100M dataset [24], achieving the state-of-the-art performance. In this

| **Ingredients** | **flour, eggs, baking soda, salt, pepper, water, shrimp, batter, breadcrumbs, oil** | | |
|---|---|---|---|
| | step 1 | step 2 | step 3 |
| Clip sequence |  | | |
| MART + Ingredients (MART-I) | add **flour salt** and **pepper** to a bowl and mix (✗ eggs, baking soda) | add milk egg and milk to the bowl and mix (✗ water) | coat the dough in the **batter** (✗ shrimp, breadcrumbs) |
| V + Ingredients (VI) | mix **flour salt pepper** and breadcrumbs (✗ baking soda, eggs) | mix flour salt pepper and breadcrumbs with the flour (✗ water) | coat the **shrimp** with the flour mixture (✗ batter, breadcrumbs) |
| VI + Visual simulator (VIV) | mix **flour eggs** and **salt** together (✗ baking soda, pepper) | add salt pepper to the eggs and mix (✗ water) | coat the **shrimp** in the **batter** (✗ breadcrumbs) |
| + VIV + Textual re-simulator (VIVT) | mix **flour eggs baking soda salt** and **pepper** and **salt** | add **water** eggs breadcrumbs to a bowl of **water** and mix | coat the **shrimp** in the **batter** (✗ breadcrumbs) |
| Ground truth | add **flour eggs baking soda salt** and **pepper** to the bowl and stir | add cold **water** to the bowl and stir | cover the **shrimp** in the **batter** and **breadcrumbs** |

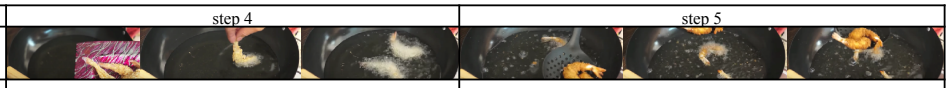
| | step 4 | step 5 |
|---|---|---|
| Clip sequence |  | |
| MART + Ingredients (MART-I) | fry the onion rings in **oil** (✗ shrimp) | remove the **shrimp** from the **oil** |
| V + Ingredients (VI) | heat **oil** in a pan and add the **shrimp** and fry | remove the **shrimp** from the **oil** |
| VI + Visual simulator (VIV) | heat **oil** in a pan and fry the **shrimp** in it | remove the **shrimp** from the **oil** |
| VIV + Textual re-simulator (VIVT) | fry the **shrimp** in **oil** | remove the **shrimp** from the **oil** |
| Ground truth | place the **shrimp** into a pan of hot **oil** | remove the **shrimp** from the pan |

**Figure 4: Examples of generated recipes. Here, we compare four models, MART-I (baseline), VI, VIV, and VIVT with the ground truth. Green bold and red words represent semantically correct and incorrect ingredients, respectively. Words in parentheses indicate missing ingredients, which should be included in the sentence. Note that parallel words in a sentence are not separated from the commas in the YouCook2 dataset (see step 1 in the ground truth).**

**Table 5: Results of retrieval evaluation. ↓ indicates that lower is better.**

| Baseline | MedR (↓) | R@1 | R@5 | R@10 |
|---|---|---|---|---|
| Transformer-XL | 214 | 1.1 | 4.9 | 9.0 |
| + Ingredients (Transformer-XL-I) | 144.5 | 1.9 | 7.0 | 11.3 |
| MART | 179 | 1.5 | 5.9 | 9.8 |
| + Ingredients (MART-I) | 103 | 2.3 | 9.5 | 14.6 |
| Ours | | | | |
| Video only (V) | 244 | 1.2 | 4.5 | 7.3 |
| V + Ingredients (VI) | 90 | 3.8 | 11.6 | 18.1 |
| VI + Visual simulator (VIV) | 67 | 4.0 | **14.3** | 20.9 |
| VIV + Textual re-simulator (VIVT) | **66** | **4.2** | 13.9 | **21.4** |
| Ground truth | 10 | 15.2 | 39.3 | 51.8 |

task, given a generated step-level sentence as a query, the MIL-NCE model embeds it and computes the cosine similarly as a score between the query vector and all the 2,493 clip vectors from the test set. Then, we sort scores with clips in descending order and calculate the median rank (MedR) and recall rate at the top $K$ (R@$K$). The median rank represents the median ranking of retrieved corresponding clips, hence lower is better; in contrast, R@$K$ represents the percentage of all the step-level sentence queries where the corresponding clip is retrieved in the top $K$, hence higher is better.

**Results.** Table 5 shows the results of the retrieval evaluation. We observe that the proposed method significantly outperforms the state-of-the-art video captioning models. In MedR, the VIVT model achieves 66, which is marginally lower than that of Transformer-XL-I 144.5 and MART-I 103. This indicates that the proposed method generates a more concrete recipe based on clips than the state-of-the-art video captioning models. In our ablation, the VIV model dramatically improves the VI model, indicating that the visual simulator is essential for generating concrete recipes. In addition, the VIVT model shows a steady improvement from the VIV model, indicating the effectiveness of the textual re-simulator.

## 4.5 Qualitative analysis

Figure 4 shows an example of the generated recipes. Other examples are presented in Appendix D.

**Insights.** For the VPC task, it is important to generate correct ingredients that are manipulated in a clip. MART-I fails to generate ingredients correctly; the model tends to miss and hallucinate ingredients (e.g., "eggs" and "milk" in steps 1 and 2). We can observe a similar tendency to the VI model, indicating that these models superfluously generate ingredients listed in the ingredient list.

The VIV model suppresses these problems (e.g., "batter" in step 3). In addition, owing to the textual re-simulator, the VIVT model can generate ingredients that are missed in the VIV model (e.g., "baking soda" and "pepper" in step 1, and "water" in step 2).

**Limitations.** Although the proposed method generates recipes more accurately than the baseline models, we still found some differences from the ground truth. For example, in step 2, the VIV

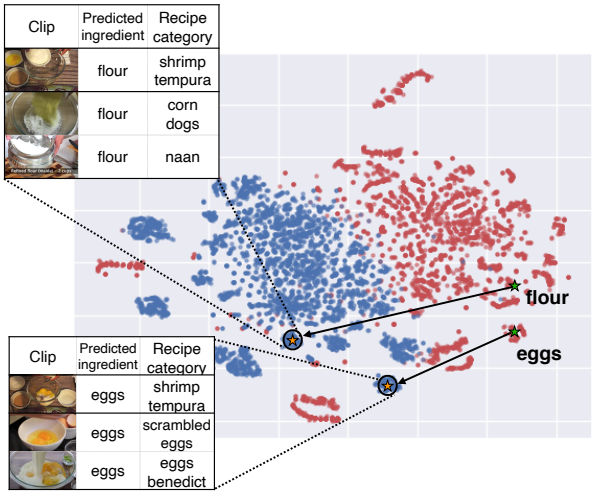



**Figure 5: Learned embedding of ingredients obtained by the VIVT model. Note that only raw and updated (the attention weight in the material selector is higher than 0.5) ingredients are transformed by t-SNE [41]. Red and blue colors represent the raw and updated ingredients, respectively.**

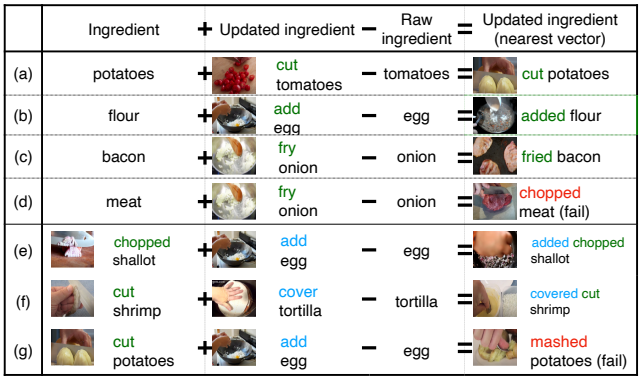



**Figure 6: Arithmetics using the learned embedding of ingredients. Examples (a) to (d) and (e) to (g) represent the first-order (raw-to-updated) and second-order (updated-to-updated) transformations, respectively. (d) and (g) show the failure cases for each transformation.**

and VIVT models refer to "salt," "pepper," "eggs," and "breadcrumbs" superfluously, but only "water" is added in the ground truth. In addition, in step 3, the VIV and VIVT models refer only to "batter," but "breadcrumbs" are also used. To solve these problems, we believe that incorporating fine-grained ingredient recognition modules [7] would help the model to generate a recipe more precisely.

## 4.6 Discussion of the learned embedding

To investigate how the visual simulator represents the state transition of ingredients, we visualize the ingredient embedding by projecting it to 2D space using t-SNE [41]. Figure 5 shows the projected learned embedding of ingredients obtained by the VIVT model. Note that only raw (red) and updated (blue)[4] ingredients are shown in this figure. This result shows that the raw and updated points are clearly divided into two main clusters in the vector space[5].

We also investigate the ingredients' trajectory with the retrieved top-2 nearest ingredient vectors from updated ingredient vectors (see the zoomed parts of the figure). The retrieved ingredients indicate their state-awareness; that is, ingredients with the similar states are embedded into the same cluster in the vector space regardless of the difference in their recipe categories defined by [47]. For example, the near vectors of updated "eggs" with the "beat" state are also updated "eggs" with "beat"-like states (e.g., mix and stir). "flour" also has the same tendency.

**Semantic vector arithmetic.** To demonstrate the state-awareness of learned embedding, we attempt to apply simple arithmetic operations as performed in the literature [25, 32, 34]. In the context of our task, the state transition of ingredients is expected to be computed as $v(\text{cut potatoes}) = v(\text{potatoes}) + v(\text{cut tomatoes}) - v(\text{tomatoes})$, where $v$ represents the map into the embedding space.

Figure 6 shows seven examples of the arithmetic operations. From (a) to (d), first-order transformations (raw-to-updated) are described, and from (e) to (g), second-order transformations (updated-to-updated) are described. We can see that the learned embedding simulates the state transition of ingredients by specific actions in both transformations. For example, in (a) and (e), "raw potatoes" and "chopped shallot" are converted into "cut potatoes" and "added chopped shallot," respectively. However, we observe some failure cases, where (d) and (g), "raw meat" is transformed into "chopped meat" via "fry" and "cut potatoes" into "mashed potatoes" via "add." In these examples, ingredients are consistent before and after, but executed actions are different because of the failure in action selection. We believe that noisy action labeling causes this failure, and a sophisticated action selection would ease this problem.

## 5 CONCLUSION

In this paper, we proposed a new VPC task for generating a procedural text from a clip sequence and material list. To generate a procedural text accurately, it is essential for models to track material states in a clip sequence. To achieve this, we proposed a novel VPC method, which modifies the existing simulator as a visual simulator and incorporates it into the encoder-decoder architecture. Our experimental results with thorough ablation demonstrate the effectiveness of the proposed method, which outperforms the state-of-the-art video captioning models. The learned embedding of materials demonstrates that the simulator effectively captures their state transition.

## ACKNOWLEDGEMENT

This work was supported by JSPS KAKENHI Grant Number JP21J20250 and JP20H04210, and partially supported by JP21H04910, JP17H06100, JST-Mirai Program Grant Number JPMJMI21G2, and JST ACT-I Grant Number JPMJPR17U5.

[4]The attention weight in the material selector was higher than 0.5.
[5]The raw and updated ingredients correspond to an embedding $\mathcal{E}^0$ by the material encoder and an embedding $\mathcal{E}^n$ updated from $\mathcal{E}^0$ by the visual simulator, respectively.

# REFERENCES


[1] Jean-Baptiste Alayrac, Piotr Bojanowski, Nishant Agrawal, Josef Sivic, Ivan Laptev, and Simon Lacoste-Julien. 2016. Unsupervised learning from narrated instruction videos. In *Proc. CVPR*. 4575–4583.

[2] Jean-Baptiste Alayrac, Josef Sivic, Ivan Laptev, and Simon Lacoste-Julien. 2017. Joint discovery of object states and manipulation actions. In *Proc. ICCV*. 2127–2136.

[3] Mustafa Sercan Amac, Semih Yagcioglu, Aykut Erdem, and Erkut Erdem. 2019. Procedural reasoning networks for understanding multimodal procedures. In *Proc. CoNLL*. 441–451.

[4] Satanjeev Banerjee and Alon Lavie. 2005. METEOR: an automatic metric for MT evaluation with improved correlation with human judgments. In *Proc. ACL Workshop IEEMMTS*. 65–72.

[5] Antoine Bosselut, Omer Levy, Ari Holtzman, Corin Ennis, Dieter Fox, and Yejin Choi. 2018. Simulating action dynamics with neural process networks. In *Proc. ICLR*.

[6] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. 2020. End-to-end object detection with transformers. In *Proc. ECCV*. 213–229.

[7] Jingjing Chen and Chong wah Ngo. 2016. Deep-based ingredient recognition for cooking recipe retrieval. In *Proc. ACMMM*. 32–41.

[8] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: attentive language models beyond a fixed-length context. In *Proc. ACL*. 2978–2988.

[9] Bhavana Dalvi, Lifu Huang, Niket Tandon, Wen tau Yih, and Peter Clark. 2018. Tracking state changes in procedural text: a challenge dataset and models for process paragraph comprehension. In *Proc. NAACL*. 1595–1604.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proc. NAACL*. 4171–4186.

[11] Jeff Donahue, Lisa Anne Hendricks, Marcus Rohrbach, Subhashini Venugopalan, Sergio Guadarrama, Kate Saenko, and Trevor Darrell. 2015. Long-term recurrent convolutional networks for visual recognition and description. In *Proc. CVPR*. 2625–2634.

[12] Victor Escorcia, Fabian Caba Heilbron, Juan Carlos Niebles, and Bernard Ghanem. 2016. DAPs: deep action proposals for action understanding. In *Proc. ECCV*. 768–784.

[13] Aditya Gupta and Greg Durrett. 2019. Tracking discrete and continuous entity state for process understanding. In *Proc. NAACL Workshop SPNLP*. 7–12.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proc. CVPR*. 770–778.

[15] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: accelerating deep network training by reducing internal covariate shift. In *Proc. ICML*. 448–456.

[16] Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical reparametrization with gumble-softmax. In *Proc. ICLR*.

[17] Jermsak Jermsurawong and Nizar Habash. 2015. Predicting the structure of cooking recipes. In *Proc. EMNLP*. 781–786.

[18] Chloé Kiddon, Ganesa Thandavam Ponnuraj, Luke Zettlemoyer, and Yejin Choi. 2015. Mise en Place: unsupervised interpretation of instructional recipes. In *Proc. EMNLP*. 982–992.

[19] Diederik P. Kingma and Jimmy Ba. [n.d.]. Adam: A method for stochastic optimization. In *Proc. ICLR*. USA.

[20] Jie Lei, Liwei Wang, Yelong Shen, Dong Yu, Tamara Berg, and Mohit Bansal. 2020. MART: memory-augmented recurrent transformer for coherent video paragraph captioning. In *Proc. ACL*. 2603–2614.

[21] Chin-Yew Lin and Franz Josef Och. 2004. Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics. In *Proc. ACL*. 605–612.

[22] Hirokuni Maeta, Tetsuro Sasada, and Shinsuke Mori. 2015. A framework for procedural text understanding. In *Proc. IWPT*. 50–60.

[23] Antoine Miech, Jean-Baptiste Alayrac, Lucas Smaira, Ivan Laptev, Josef Sivic, and Andrew Zisserman. 2020. End-to-end learning of visual representations from uncurated instructional videos. In *Proc. CVPR*. 9879–9889.

[24] Antoine Miech, Dimitri Zhukov, Jean-Baptiste Alayrac, Makarand Tapaswi, Ivan Laptev, and Josef Sivic. 2019. HowTo100M: learning a text-video embedding by watching hundred million narrated video clips. In *Proc. ICCV*. 2630–2640.

[25] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NeurIPS*. 3111–3119.

[26] Mike Mintz, Steven Bills, Rion Snow, and Daniel Jurafsky. 2009. Distant supervision for relation extraction without labeled data. In *Proc. ACL-IJCNLP*. 1003–1011.

[27] Taichi Nishimura, Atsushi Hashimoto, Yoshitaka Ushiku, Hirotaka Kameko, Yoko Yamakata, and Shinsuke Mori. 2020. Structure-aware procedural text generation from an image sequence. *IEEE Access* 9 (2020), 2125–2141.

[28] Liangming Pan, Jingjing Chen, Jianlong Wu, Shaoteng Liu, Chong-Wah Ngo, Min-Yen Kan, Yu-Gang Jiang, and Tat-Seng Chua. 2020. Multi-modal cooking workflow construction for food recipes. In *Proc. ACMMM*. 1132–1141.

[29] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proc. ACL*. 311–318.

[30] Jae Sung Park, Marcus Rohrbach, Trevor Darrell, and Anna Rohrbach. 2019. Adversarial inference for multi-sentence video description. In *Proc. CVPR*. 6598–6608.

[31] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: global vectors for word representation. In *Proc. EMNLP*. 1532–1543.

[32] Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv.

[33] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: towards real-time object detection with region proposal networks. In *Proc. NeurIPS*. 91–99.

[34] Amaia Salvador, Nicholas Hynes, Yusuf Aytar, Javier Marin, Ferda Ofli, Ingmar Weber, and Antonio Torralba. 2017. Learning cross-modal embeddings for cooking recipes and food images. In *Proc. CVPR*. 3020–3028.

[35] Adam Santoro, Ryan Faulkner, David Raposo, Jack Rae, Mike Chrzanowski, Theophane Weber, Daan Wierstra, Oriol Vinyals, Razvan Pascanu, and Timothy Lillicrap. 2019. Relational recurrent neural networks. In *Proc. NeurIPS*. 7299–7310.

[36] Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: summarization with pointer-generator networks. In *Proc. ACL*. 1073–1083.

[37] Botian Shi, Lei Ji, Yaobo Liang, Nan Duan, Peng Chen, Zhendong Niu, and Ming Zhou. 2019. Dense procedure captioning in narrated instructional videos. In *Proc. ACL*. 6382–6391.

[38] Botian Shi, Lei Ji, Zhendong Niu, Nan Duan, Ming Zhou, and Xilin Chen. 2020. Learning semantic concepts and temporal alignment for narrated video procedural captioning. In *Proc. ACMMM*. 4355–4363.

[39] Chen Sun, Austin Myers, Carl Vondrick, Kevin Murphy, and Cordelia Schmid. 2019. VideoBERT: a joint model for video and language representation learning. In *Proc. ICCV*. 7464–7473.

[40] Ganchao Tan, Daqing Liu, Meng Wang, and Zheng-Jun Zha. 2020. Learning to discretely compose reasoning module networks for video captioning. In *Proc. IJCAI*. 745–752.

[41] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research* 9 (2008), 2579–2605.

[42] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proc. NeurIPS*. 5998–6008.

[43] Ramakrishna Vedantam, C. Lawrence Zitnick, and Devi Parikh. 2015. CIDEr: consensus-based image description evaluation. In *Proc. CVPR*. 4566–4575.

[44] Yilei Xiong, Bo Dai, and Dahua Lin. 2018. Move forward and tell: a progressive generator of video descriptions. In *Proc. ECCV*. 489–505.

[45] Nadav Zamir, Asaf Noy, Itamar Friedman, Matan Protter, and Lihi Zelnik-Manor. 2020. Asymmetric loss for multi-label classification. arXiv.

[46] Luowei Zhou, Yannis Kalantidis, Xinlei Chen, Jason J. Corso, and Marcus Rohrbach. 2019. Grounded video description. In *Proc. CVPR*. 6578–6587.

[47] Luowei Zhou, Chenliang Xu, and Jason J. Corso. 2018. Towards automatic learning of procedures from web instructional videos. In *Proc. AAAI*. 7590–7598.

[48] Luowei Zhou, Yingbo Zhou, Jason J. Corso, Richard Socher, and Caiming Xiong. 2018. End-to-end dense video captioning with masked transformer. In *Proc. CVPR*. 8739–8748.