State-aware Video Procedural Captioning

Taichi Nishimura[1*], Atsushi Hashimoto[2], Yoshitaka Ushiku[2], Hirotaka Kameko[3] and Shinsuke Mori[3]

[1*]Graduate School of Informatics, Kyoto University, Yoshidahonmachi, Kyoto-shi, 606–8501, Kyoto, Japan.
[2]OMRON SINIC X Corporation, 5–24–5, Bunkyo-ku, 113–8656, Tokyo, Japan.
[3]Academic Center for Computing and Media Studies, Kyoto University, Yoshidahonmachi, Kyoto-shi, 606–8501, Kyoto, Japan.

*Corresponding author(s). E-mail(s): taichitary@gmail.com;
Contributing authors: atsushi.hashimoto@sinicx.com;
yoshitaka.ushiku@sinicx.com; kameko@i.kyoto-u.ac.jp;
forest@i.kyoto-u.ac.jp;

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 set. 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.

Keywords: Instructional video, Procedural text, Simulator

1 Introduction

In recent years, there has been significant progress in vision and language research that targets procedural text and instructional video [1–5]. 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 [6, 7] (VPC), which is the task of generating procedural text from instructional videos, has been proposed. VPC requires a model (1) to segment important clips from a video, (2) enumerate materials used in the clips, and (3) 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 [1, 8] and object recognition [9, 10], respectively. In this paper, we focus on (3) by designing a new VPC task and the method of generating a procedural text from a clip sequence pre-segmented in an instructional video and material set (Fig. 1).

Different from standard video captioning [11–15], 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 Fig. 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 Fig. 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., 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 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 [16, 17]. These simulators read one sentence in a procedural text, predict executed actions and involved materials, and recurrently update the state of the 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. Fig. 1 illustrates the idea of the proposed method. Given a clip sequence and material set, the proposed method reasons the state transition of materials and generates 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.

Key contributions and extensions from our previous work. The preliminary version of our work has been published at ACM Multimedia 2021 [18]. The key contribution of [18] is (1) modifying an existing NLU simulator as a visual simulator and incorporating it into an encoder-decoder architecture, and (2) incorporating textual re-simulator to facilitate the model to generate a procedural text more accurately. Furthermore, this work provides a detailed explanation of the proposed method and additional experiments as following:

- Modification of the material encoder for learning material *set* of permutation invariant representations of materials. Although the previous work regarded input materials as an ordered set, it is more reasonable to generate identical sentences against changes in the order of materials.

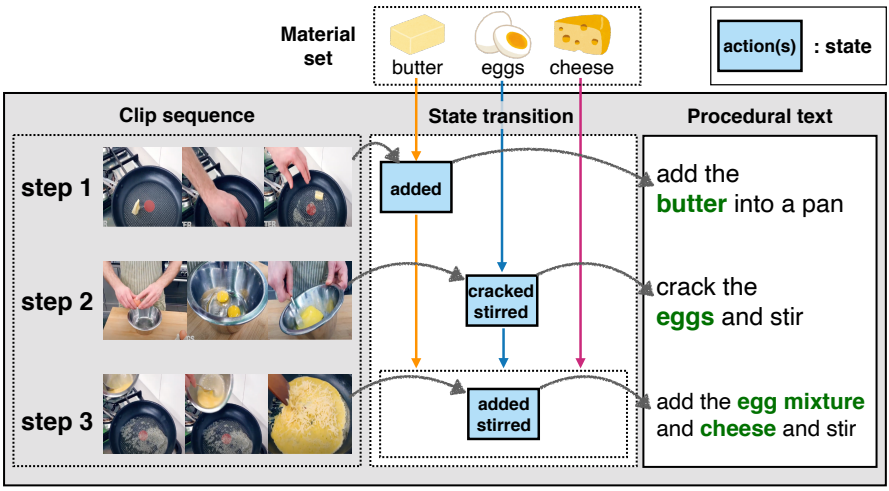

Fig. 1 Concept of the proposed method. Given a clip sequence and material list, the proposed method generates a procedural text by reasoning the state transition of materials at each step.

Table 1 A comparison of video captioning approaches. Our work is the first attempt to generate a procedural text more accurately than other general video captioning methods without using transcription.

Method	Input	Domain	Base model
MFT [14]	Video	General	LSTM
AdvInf [19]	Video	General	LSTM
Masked Transformer [20]	Video	General	Transformer
Transformer-XL [13]	Video	General	Transformer
MART [13]	Video	General	Transformer
DPC [6]	Video + Transcription	Procedural	LSTM
TVPC [7]	Video + Transcription	Procedural	Transformer
Ours	Video + Materials	Procedural	Transformer

- Increase of the dataset size by annotating new examples. We added 126 newly annotated recipes to our previous study.
- Additional quantitative and qualitative evaluations, such as sentence-level word-overlap evaluation and additional examples of generated recipes.
- Quantitative evaluation of the visual simulators to investigate how accurately they can acquire the state-awareness of materials.
- Additional experiments on the full prediction setting, where the material set is not given but is predicted from the video clips in advance. These experiments provide insight into whether the proposed methods work well with the predicted ingredients.

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. Table 1 shows a comparison of recent video captioning approaches. 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 [13, 14, 19], which aims to generate multiple sentences for a given clip sequence pre-segmented in the video.

In the video paragraph description, recently, Transformer-based [21] models have been featured because of their ability to capture long-term information in a clip sequence, compared with LSTM-based models [14, 19]. MART [13] 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.

Table 2 A comparison of methods for procedural text understanding. This work is the first attempt to apply a reasoning-based simulator to the video captioning task.

Method	Type	Modality	Task
Action graph [22]	Graph-based parser	Text	Parsing
Recipe flow graph [23]	Graph-based parser	Text	Parsing
SIMMR [24]	Graph-based parser	Text	Parsing
MM-ReS [25]	Graph-based parser	Image+Text	Parsing
vSIMMR [26]	Graph-based parser	Image+Text	Parsing
NPN [16]	Reasoning-based simulator	Text	Text generation
RPN [17]	Reasoning-based simulator	Image+Text	QA
Ours	Reasoning-based simulator	Video+Text	Video captioning

Different from standard video captioning, VPC requires a model to generate detailed material manipulations for input clips. To this end, previous VPC works [6, 7] 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 set[1]. 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. Table 2 shows a comparison of these approaches.

A graph-based parser aims to train a model to map recipes into graph or tree structures. Kiddon *et al.* [22] and Maeta *et al.* [23] proposed a model to estimate a graph structure called the action graph and recipe flow graph, respectively. Jermsurawong and Nizar [24] proposed a parser with the SIMMR dataset, which provides a tree structure for understanding ingredient merging operations. Then, Pan *et al.* [25] and Nishimura *et al.* [26] 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

[1]We perform an experiments on the full prediction setting in Section 4.7.

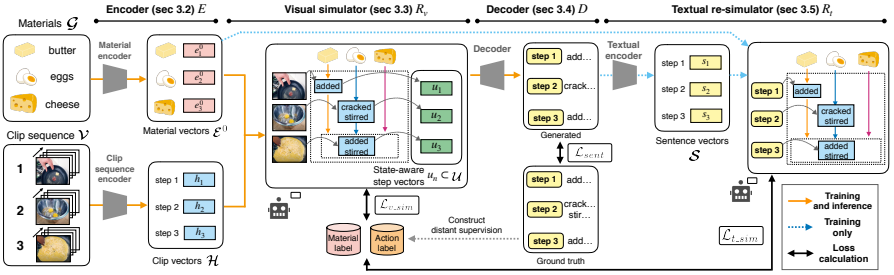

Fig. 2 An overview of the proposed method. To track material states in a clip sequence, we incorporate the visual simulator R_v into the transformer-based encoder-decoder architecture (E and D). In addition, based on our intuition that the state transition of materials is traceable from the generated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

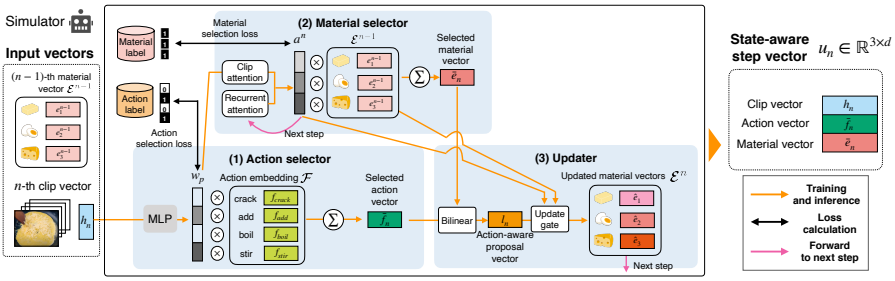

Fig. 3 An overview of the (visual) simulator. The simulator recurrently reasons the state transition of the materials at each step. Specifically, it predicts executed actions and involved materials in (1) the action and (2) material selector and then updates the state of materials in (3) the updater.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

Durrett [27] 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.* [17] proposed a method to answer multi-modal question answering (QA) tasks by reasoning the state transition of ingredients using a relation reasoning network [28]. This paper builds upon the neural process network (NPN) [16], 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 [17, 29]. 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 method from Section

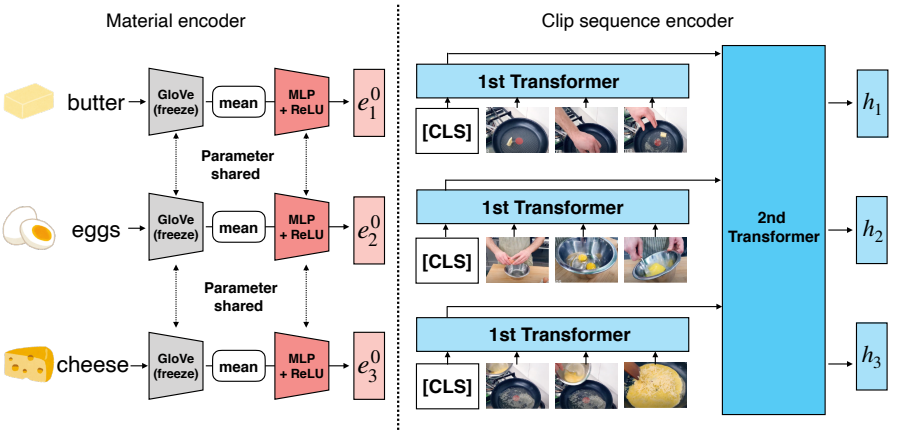

Fig. 4 An overview of the material and clip sequence encoders.

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

3.1 Overview

Fig. 2 shows an overview of the proposed method. Given a clip sequence $\boldsymbol{\mathcal{V}} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n, \ldots, \boldsymbol{v}_N)$ and a material set $\boldsymbol{\mathcal{G}} = (\boldsymbol{g}_1, \ldots, \boldsymbol{g}_m, \ldots, \boldsymbol{g}_M)$, our goal is to output a procedural text $\boldsymbol{\mathcal{Y}} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n, \ldots, \boldsymbol{y}_N)$ 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 $\boldsymbol{\mathcal{H}}$ and material encoded vectors $\boldsymbol{\mathcal{E}}^0$, the visual simulator reasons the state transition of materials and outputs state-aware step vectors as $\boldsymbol{\mathcal{U}} = R_v(\boldsymbol{\mathcal{H}}, \boldsymbol{\mathcal{E}}^0)$. Then, the decoder D outputs a procedural text conditioned on $\boldsymbol{\mathcal{U}}$. 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 $R_t(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{E}}^0)$, where $\boldsymbol{\mathcal{S}}$ 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 set $\mathcal{G}$ and a clip sequence $\boldsymbol{\mathcal{V}}$. Thus, we develop a suitable encoder for each component. Fig. 4 shows an overview of the encoders.

Material encoder. To encode a material set, we input them to concatenated neural networks of pre-trained GloVe [30][2] 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. Then, we obtain the initial material vectors $\boldsymbol{\mathcal{E}}^0 = (\boldsymbol{e}_1^0, \ldots, \boldsymbol{e}_m^0, \ldots, \boldsymbol{e}_M^0)$. Compared with our previous work, the key difference is the learning of permutation-invariant representations of materials instead of the position-sensitive implementation in the previous work. Although the previous work regarded input materials as ordered sets, it is more reasonable to generate identical sentences against changes in the order of materials.

Clip sequence encoder. A clip sequence $\boldsymbol{\mathcal{V}}$ 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 [13, 31, 32]. Then the latter transformer is trained over the sequence to obtain the step-aware clip vectors $\boldsymbol{\mathcal{H}} = (\boldsymbol{h}_1, \ldots, \boldsymbol{h}_n, \ldots, \boldsymbol{h}_N)$ in the clip sequence.

3.3 Visual simulator

Based on the encoded vectors of the clip sequence and material set $(\boldsymbol{\mathcal{H}}, \boldsymbol{\mathcal{E}}^0)$, the visual simulator, shown in Fig. 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 set $\boldsymbol{\mathcal{E}}^{n-1}$, 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 $\boldsymbol{\mathcal{H}}$ and material vectors $\boldsymbol{\mathcal{E}}^{\mathbf{0}}$ 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 $\boldsymbol{\mathcal{F}}$. For example, in Fig. 3, the actions "crack" and "stir" are executed in the clip, thus both $\boldsymbol{f}_{crack}$ and $\boldsymbol{f}_{stir}$ should be selected.

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

To consider multiple actions, the action selector computes a soft selection $\boldsymbol{w}_p$ as an action probability for each action in $\boldsymbol{\mathcal{F}}$. Then it outputs the selected action vector $\bar{\boldsymbol{f}}_n$ as a weighted sum of the action embedding $\boldsymbol{\mathcal{F}}$ 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 set $\boldsymbol{\mathcal{E}}^{n-1}$. For example, in Fig. 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 set $\boldsymbol{\mathcal{E}}^{n-1}$. Then it outputs the selected material vector $\bar{\boldsymbol{e}}_n$ as a weighted sum of the material vectors $\boldsymbol{\mathcal{E}}^{n-1}$ 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 $\boldsymbol{e}_m^{n-1}$. The m-th new material vector $\hat{\boldsymbol{e}}_m$ is assigned to $\boldsymbol{\mathcal{E}}_m^n$, which is forwarded to the next $(n+1)$-th step.

3.4 Decoder

Fig. 5 shows an overview of the 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 [6, 7, 20]. Our task allows the model to refer to a material set, thus to encourage the decoder to generate materials, we incorporate the copy mechanism [33] into the decoder D. When generating the k-th word in the n-th sentence, given the updated materials $\boldsymbol{e}_m^n \in \boldsymbol{\mathcal{E}}^n$, the copy mechanism first calculates the attention probability $\boldsymbol{\beta}_{n,k}^m$ using the bilinear dot product of the vectors of the decoder output $\boldsymbol{o}_{n,k}$ and each material $\boldsymbol{e}_m^n \in \boldsymbol{\mathcal{E}}^n$ as:

$$\boldsymbol{\beta}_{n,k}^m = \frac{\exp\{(\boldsymbol{o}_{n,k})^\mathsf{T} \boldsymbol{W}_c \boldsymbol{e}_m^n\}}{\sum_i \exp\{(\boldsymbol{o}_{n,k})^\mathsf{T} \boldsymbol{W}_c \boldsymbol{e}_i^n\}}, \tag{1}$$

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

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

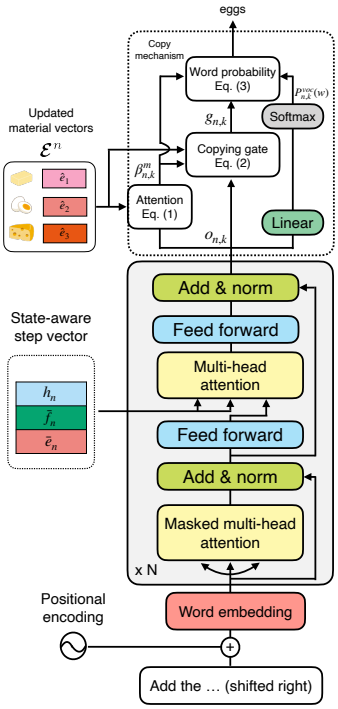

Fig. 5 An overview of the decoder.

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:

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

where $P^{\mathrm{voc}}_{n,k}(w)$ 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[3].

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

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.

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 $\boldsymbol{\mathcal{S}} = (\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n, \ldots, \boldsymbol{s}_N)$. First, it applies the straight-through version of Gumbel softmax resampling [34] 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 $\boldsymbol{\mathcal{S}}$ using a biLSTM encoder. Finally, based on $\boldsymbol{\mathcal{S}}$, the textual simulator, another NPN described in Section 3.3, reasons the state transition of materials again as $R_t(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{E}}^0)$.

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 [35] following the original NPN training method [16]. For the material selection loss, labels are obtained whether or not each step contains materials in the material set; for the action selection loss, labels are obtained whether or not each step contains actions in the 384 actions defined by [16]. 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 [36], 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.

Table 3 YouCook2-ingredient+ dataset statistics.

	train	val	test
#recipes	1,331	228	229
#steps / #recipes	7.8	7.6	7.7
#ingredients / #recipes	10.4	10.3	10.4

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 λ 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).

4.1 Experimental settings

Dataset. We use the YouCook2 dataset [1], 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. Because ingredients are not annotated in the original dataset, in our previous work, we prepared the YouCook2-ingredient dataset by annotating ingredients for recipes with 1,662 valid videos that are available online. This work increases the dataset size by obtaining the missing videos through YouCook2 author and annotating ingredients with these additional videos. As a result, we obtained the annotated 1,788 videos as the YouCook2-ingredient+ dataset and divided them as follows: 1,331 for training, 228 for validation, and 229 for testing[4]. Table 3 shows the statistics of the YouCook2-ingredient+ dataset.

Data preprocessing. As clip features, we use concatenated features of appearance and optical flow provided by [1]. With respect to the appearance, 2,048D feature vectors extracted from the "Flatten-673" layer in ResNet-200 [37] are used, and for the optical flow, 1,024D feature vectors extracted from the "global pool" layer in BN-Inception [38] are used. As in [13], we truncated sequences longer than 100 for the clip and 20 for the sentence and set the

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

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 991 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 [13, 31]; we use the Adam optimizer [39] 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 λ with four different values $\lambda \in \{0.25, 0.5, 0.75, 1.0\}$ and set λ 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** [40] is a powerful transformer-based language model that was originally proposed for capturing long-term dependency in natural language. As in [13], we adapt it for our task; the model directly uses all of the previous hidden states to generate a current sentence.
- **MART** [13] 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 summarize important information in the previous step.

Note that these models originally have no ingredient set 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 [41], ROUGE-L [42], METEOR [43], and CIDEr-D [44] in the test set. We conduct two

Table 4 Paragraph- and sentence-level word-overlap evaluation 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.

		Paragraph-level							Sentence-level						
Baseline	I	B1	B2	B3	B4	M	C	RL	B1	B2	B3	B4	M	C	RL
Transformer-XL		39.0	22.0	12.1	6.7	15.2	22.7	30.9	26.5	13.5	5.3	1.5	10.6	57.3	27.8
+ Ingredients (Transformer-XL-I)	✓	37.7	22.5	13.4	8.2	15.4	35.4	34.2	29.6	17.0	8.2	3.5	12.6	69.6	31.3
MART		37.9	21.7	12.4	7.6	15.0	29.1	32.3	28.0	14.5	6.1	2.4	11.1	62.1	29.4
+ Ingredients (MART-I)	✓	42.3	26.2	16.1	9.9	17.6	48.2	36.2	31.3	18.5	9.4	4.4	13.7	81.0	32.9
Ours															
Video only (V)		43.2	24.5	14.0	8.1	16.6	32.4	31.9	28.0	13.9	6.0	1.9	11.4	60.7	28.3
V + Ingredients (VI)	✓	49.1	29.5	17.6	10.5	20.3	63.3	35.2	31.9	18.0	9.3	3.8	13.9	81.7	31.3
VI + Visual simulator (VIV)	✓	**49.4**	30.1	18.0	11.0	21.0	66.1	36.8	**33.7**	19.3	9.7	4.4	**15.2**	93.0	33.3
VIV + Textual re-simulator (VIVT)	✓	**49.4**	**30.9**	**18.3**	**11.3**	**21.1**	**67.1**	**37.1**	33.5	**19.4**	**10.1**	**4.9**	**15.2**	**96.7**	**33.7**

Table 5 Change in paragraph-level word-overlap evaluation with controlled λ.

	B1	B2	B3	B4	M	C	RL
$\lambda = 0$ (VIV)	49.4	30.1	18.0	11.0	21.0	66.1	36.8
$\lambda = 0.25$	49.6	30.4	**18.3**	**11.3**	21.1	65.5	36.6
$\lambda = 0.5$	49.4	**30.9**	**18.3**	**11.3**	21.1	**67.1**	**37.1**
$\lambda = 0.75$	**50.3**	30.4	18.0	10.8	**21.2**	65.7	36.3
$\lambda = 1.0$	49.1	30.0	18.0	11.0	21.0	64.4	36.7

types of word-overlap evaluation: recipe (paragraph-level) [13, 14, 19] and step (sentence)-level evaluation [7].

Results. Table 4 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 in both paragraph- and sentence-level evaluation. 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 λ. Table 5 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 achieves a performance that is better or comparable to the VIV model for any λ values, and (2) $\lambda = 0.5$ performs the best among the three metrics (BLEU2-4, ROUGE-L, and CIDEr-D) and obtains competitive results in BLEU1 and METEOR. 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.

(a)

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 pepper** to a bowl (✗ baking soda, egg)	add flour salt pepper and milk to the bowl and mix (✗ water)	add flour and **shrimp** in the bowl (✗ batter, breadcrumbs)
V + Ingredients (VI)	mix **flour salt pepper** and breadcrumbs (✗ baking soda, eggs)	mix flour salt pepper and breadcrumbs (✗ water)	coat the **shrimp** in the batter (✗ batter, breadcrumbs)
VI + Visual simulator (VIV)	mix **flour baking soda salt pepper** and breadcrumbs (✗ eggs)	mix the eggs with eggs (✗ water)	coat the **shrimp** in the **batter** (✗ breadcrumbs)
+ VIV + Textual re-simulator (VIVT)	mix **flour baking soda salt** and **pepper** together (✗ eggs)	mix the eggs with the **water**	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 **shrimp** in **oil**	remove the **shrimp** from the **oil**
V + Ingredients (VI)	fry the **shrimp** in a pan with **oil**	remove the **shrimp** from the **oil**
VI + Visual simulator (VIV)	heat **oil** in a pan and fry the **shrimp**	remove the **shrimp** from the **oil**
VIV + Textual re-simulator (VIVT)	heat **oil** in a pan and fry the **shrimp**	remove the **shrimp** from the **oil**
Ground truth	place the **shrimp** into a pan of hot **oil**	remove the **shrimp** from the pan

(b)

Ingredients	**soy sauce, brown sugar, water, garlic, green onions, sesame oil, ribs**		
	step 1	step 2	step 3
Clip sequence			
MART + Ingredients (MART-I)	add **soy sauce sesame oil soy sauce sesame oil** and **sesame oil** to a pan (✗ brown sugar, garlic, green onions, water)	cut the **ribs** into small pieces	add soy sauce sesame oil soy sauce sesame oil and sesame oil to a bowl (✗ marinade)
V + Ingredients (VI)	add **chopped garlic sesame oil brown sugar** and red pepper flakes to a pan and mix (✗ green onions, water)	put the marinade in the **ribs** and soy sauce	add sesame oil garlic and the bowl (✗ marinade)
VI + Visual simulator (VIV)	add **garlic soy sauce sugar garlic green onions** and **garlic** to a bowl and mix (✗ sesame oil, water)	cover the **ribs** with plastic wrap and place in a paper towel	pour the **marinade** over the ribs
+ VIV + Textual re-simulator (VIVT)	add **garlic sesame oil sugar soy sauce garlic** and **water** to a blender (✗ green onions)	place the **ribs** in a bag	put the ribs into a bag (✗ marinade)
Ground truth	combine **soy sauce brown sugar water garlic green onions** and **sesame oil**	place the **ribs** in the bag	pour the **marinade** into the bag

	step 4	step 5
Clip sequence		
MART + Ingredients (MART-I)	cook the ribs in the pan	cook the **ribs** on a grill
V + Ingredients (VI)	flip the pancakes over	place the **ribs** on the grill
VI + Visual simulator (VIV)	flip the ribs over	cook the **ribs** on a grill
VIV + Textual re-simulator (VIVT)	grill the ribs	place the **ribs** on the grill and cook them until it is golden brow on both sides
Ground truth	oil the grate of the grill	cook the **ribs** on the grill

Fig. 6 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 (a) in the ground truth).

Table 6 Results of ingredient prediction.

Baseline	Recall	Precision	F1
Transformer-XL	12.6	19.1	15.2
+ Ingredients (Transformer-XL-I)	17.3	30.4	22.0
MART	11.3	20.2	14.5
+ Ingredients (MART-I)	21.9	34.0	26.7
Ours			
Video only (V)	13.5	19.8	16.0
V + Ingredients (VI)	24.3	33.1	28.1
VI + Visual simulator (VIV)	28.9	**43.2**	34.7
VIV + Textual re-simulator (VIVT)	**29.7**	**43.2**	**35.2**

Table 7 Results of retrieval evaluation. ↓ indicates that lower is better.

Baseline	MedR (↓)	R@1	R@5	R@10
Transformer-XL	162.5	2.0	6.3	11.0
+ Ingredients (Transformer-XL-I)	139	1.6	8.1	13.6
MART	138.5	1.9	7.1	11.4
+ Ingredients (MART-I)	79	3.1	11.9	19.4
Ours				
Video only (V)	134	2.2	8.4	13.6
V + Ingredients (VI)	65	4.6	14.5	21.2
VI + Visual simulator (VIV)	**49**	4.6	**17.3**	25.1
VIV + Textual re-simulator (VIVT)	**49**	**5.2**	**17.3**	**25.2**
Ground truth	7	19.0	45.2	59.7

Results. Table 6 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 6.6% 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 0.5% 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 [5] pre-trained on the HowTo100M dataset [4], achieving the state-of-the-art performance. In this 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 1,768 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 a 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 a higher is better.

Results. Table 7 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 49, which is marginally

Table 8 Quantitative evaluation of the visual simulators.

	Ingredient			Action		
	Recall	Precision	F1	Recall	Precision	F1
VIV	22.6	77.5	35.0	49.5	18.7	27.1
VIVT	21.4	75.9	33.4	49.1	19.9	28.3
Distant supervision	30.5	97.8	46.5	94.7	57.1	71.2

lower than that of Transformer-XL-I 139 and MART-I 79. 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

Fig. 6 shows two examples of the generated recipes.

Insights. For the VPC task, it is important to generate the 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., "flour" and "milk" in step 2 (a) and "garlic" in step 1 (b)). A similar tendency can be observed in the VI model, indicating that these models superfluously generate ingredients listed in the ingredient set.

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

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 in (a), the VIV and VIVT models refer to "eggs," superfluously, but only "water" is added to the ground truth. In addition, in step 3 in (b), the VIV and VIVT models generate "ribs" but it is not used. To solve these problems, we believe that incorporating fine-grained ingredient recognition modules [45] would help the model to generate a recipe more precisely.

4.6 Discussion of the learned embedding

4.6.1 Quantitative evaluation of visual simulators

We evaluate the visual simulators by measuring the performance of the action and material selectors. These models are trained from distant supervision, as described in Section 3.6, which does not always agree with human judgment. Thus, we manually annotated the action and material labels for 50 recipes that were randomly sampled from the test set.

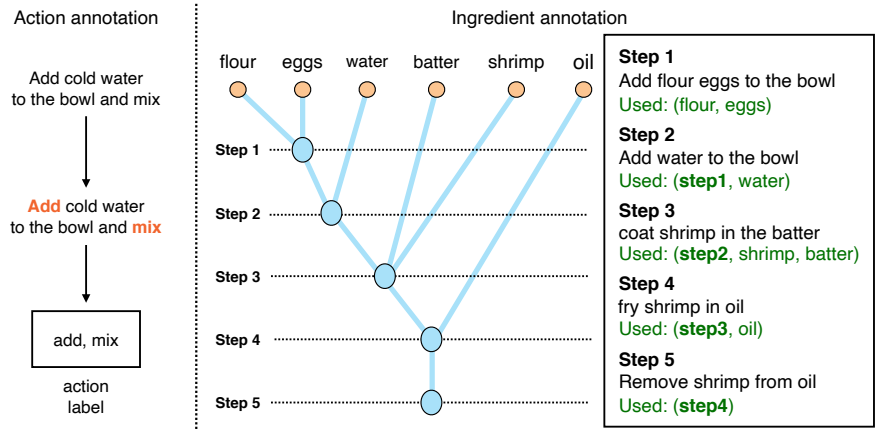

Fig. 7 An annotation example of the visual simulator.

Fig. 7 shows an annotation example. For actions, we manually extracted action words from sentences at each step (e.g., "add" and "mix" were extracted in Fig. 7). For ingredients, step sentences do not explicitly include all of the ingredients manipulated in the real world because they mention only the difference from the previous steps. For example, in step 2 in Fig. 7, a mixture of "flour" and "eggs" is used, but it is not mentioned in the sentence. To address this, we annotated the ingredient tree structure [24, 26], where each step node indicates a mixture of ingredients. The tree structure allows us to determine which ingredients are used at each step by back-tracking its leaf nodes.

Results. We compute micro-recall, precision, and F1 as evaluation metrics for the visual simulator. Table 8 shows that VIV and VIVT achieve competitive results; while VIV performs better in terms of ingredient selection, VIVT performs better with respect to action selection. We also note that even distant supervision performs only F1=46.5% in the ingredient selection. This happens because recipes mention only the difference from the previous steps; "flour" and "eggs" are manipulated in step 2 in Fig. 7, but are not mentioned. Therefore, distant supervision is not perfect for simulating human material manipulation in the real world. However, it effectively guides the model to predict incremental ingredients and is adequate for improving the captioning performance of the VPC task.

4.6.2 Visualization 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 [46]. Fig. 8 shows the projected learned embedding of ingredients obtained by the VIVT model. Note that only raw (red) and updated (blue)[5] ingredients are shown in the figure. This result shows that the raw and

[5]The attention weight in the material selector was higher than 0.5.

Table 9 Paragraph- and sentence-level word-overlap evaluation on the full prediction setting.

	I	B1	B2	B3	B4	M	C	RL	B1	B2	B3	B4	M	C	RL
Video only (V)		43.2	24.5	14.0	8.1	16.6	32.4	31.9	28.0	13.9	6.0	1.9	11.4	60.7	28.3
w/ Predict ingredients															
V + Ingredients (VI)	✓	42.5	24.3	13.9	8.1	16.5	28.4	32.3	28.5	14.6	5.9	2.2	11.8	62.1	28.7
VI + Visual simulator (VIV)	✓	42.5	24.3	13.9	8.1	16.7	33.9	32.5	28.9	14.6	6.2	2.3	11.9	63.8	28.8
VIV + Textual re-simulator (VIVT)	✓	42.7	24.0	13.3	7.7	16.7	31.6	31.9	28.3	14.0	5.5	2.3	11.7	60.8	28.3
w/ Ground-truth ingredients (Table 4)															
V + Ingredients (VI)	✓	49.1	29.5	17.6	10.5	20.3	63.3	35.2	31.9	18.0	9.3	3.8	13.9	81.7	31.3
VI + Visual simulator (VIV)	✓	49.4	30.1	18.0	11.0	21.0	66.1	36.8	33.7	19.3	9.7	4.4	15.2	93.0	33.3
VIV + Textual re-simulator (VIVT)	✓	49.4	30.9	18.3	11.3	21.1	67.1	37.1	33.5	19.4	10.1	4.9	15.2	96.7	33.7

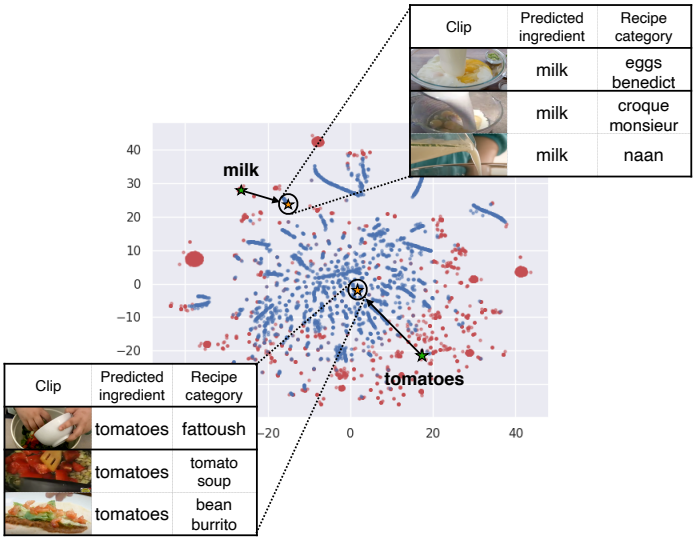

Fig. 8 Learned embedding of ingredients obtained by the VIVT model. Note that only raw and updated (the attention weight in the material selector is higher than 0.5) ingredients are transformed by t-SNE [46]. Red and blue colors represent the raw and updated ingredients, respectively.

updated points are located on the outer and inner sides of the distribution, respectively[6].

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 similar states are embedded into the same cluster in the vector space regardless of the difference in their recipe categories defined by [1]. For example, the near vectors of updated "milk" with the "add" state are also updated "milk" with "add"-like states (e.g., mix and pour). "tomatoes" also have the same tendency.

	Ingredient	+	Updated ingredient	−	Raw ingredient	=	Updated ingredient (nearest vector)
(a)	canadian bacon	+	 cut tuna	−	tuna	=	 cut canadian bacon
(b)	lettuce	+	 add tomatoes	−	tomatoes	=	 added lettuce
(c)	beef	+	 fry chicken	−	chicken	=	 fried beef
(d)	chicken	+	 cut tuna	−	tuna	=	 added chicken (fail)
(e)	 drained spaghetti	+	 mix rice	−	rice	=	 mixed drained spaghetti
(f)	 cut dogs	+	 fry chicken	−	chicken	=	 fried cut dogs
(g)	 cut potatoes	+	 fry chicken	−	chicken	=	 mixed potatoes (fail)

Fig. 9 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.

4.6.3 Semantic vector arithmetic

To demonstrate the state-awareness of learned embedding, we attempt to apply simple arithmetic operations as performed in the literature [47–49]. 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 in the embedding space.

Fig. 9 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), "canadian bacon" and "drained spaghetti" are converted into "cut canadian bacon" and "mixed drained spaghetti," respectively. However, we observe some failure cases, where (d) and (g), "raw chicken" is transformed into "added chicken" via "cut" and "cut potatoes" into "mixed potatoes" via "fry." 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.

Table 10 Results of ingredient prediction on the full prediction setting.

	Recall	Precision	F1
Video only (V)	13.5	19.8	16.0
w/ Predict ingredients			
V + Ingredients (VI)	14.8	21.5	17.5
VI + Visual simulator (VIV)	16.2	22.5	18.8
VIV + Textual re-simulator (VIVT)	15.8	21.9	18.3
w/ Ground-truth ingredients (Table 6)			
V + Ingredients (VI)	24.3	33.1	28.1
VI + Visual simulator (VIV)	28.9	43.2	34.7
VIV + Textual re-simulator (VIVT)	29.7	43.2	35.2

Table 11 Results of retrieval evaluation on the full prediction setting.

	MedR (↓)	R@1	R@5	R@10
Video only (V)	134	2.2	8.4	13.6
w/ Predict ingredients				
V + Ingredients (VI)	138.5	2.1	7.8	13.1
VI + Visual simulator (VIV)	133.5	1.6	7.2	12.1
VIV + Textual re-simulator (VIVT)	141	2.0	7.0	12.6
w/ Ground-truth ingredients				
V + Ingredients (VI)	65	4.6	14.5	21.2
VI + Visual simulator (VIV)	49	4.6	17.3	25.1
VIV + Textual re-simulator (VIVT)	49	5.2	17.3	25.2

Table 12 Performance of an ingredient decoder on the full prediction setting. Note that we compute the micro-recall, precision, and F1 on the multi-label classification setting.

	Recall	Precision	F1
VI	45.2	27.2	34.0
VI + Visual simulator (VIV)	46.0	27.8	34.7
VIV + Textual re-simulator (VIVT)	45.3	27.3	34.1

4.7 Experiments on the full prediction setting

This work discusses the proposed method under the condition that the input has ground-truth ingredients. Then, a natural question arises: *is the proposed method effective for VPC using the predicted ingredients?* To answer this question, we conduct experiments on the full prediction setting, where the material set is not given, but is predicted from the video clips in advance. To achieve this, we add an ingredient decoder of the multi-label classifier and train the entire model as multi-task learning (for details, see Appendix D).

Results. We conduct three types of evaluations: word-overlap evaluation, ingredient prediction, and retrieval evaluation. Table 9, 10, and 11 shows their results. Compared with the video-only (V) model, the performance of VI, VIV, and VIVT models is competitive in word-overlap evaluation and ingredient prediction, and is worse with respect to retrieval evaluation. Compared with the models with ground-truth ingredients, we observe a performance drop by a significant margin. This performance degradation likely occurs because of the accumulated errors of the ingredient decoder and visual simulators; Table 12 indicates that the performance of the ingredient decoder is around F1=34% to 35% on average. Therefore, we conclude that the proposed method performs

[6]The raw and updated ingredients correspond to an embedding $\boldsymbol{\mathcal{E}}^0$ by the material encoder and an embedding $\boldsymbol{\mathcal{E}}^n$ updated from $\boldsymbol{\mathcal{E}}^0$ by the visual simulator, respectively.

well with the input of ground-truth ingredients, rather than the predicted ingredients. This is another limitation of our work.

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.

Future work. We have two future directions: applying the proposed method to harder settings and extending the scope of domains. Currently, our method performs well on edited and segmented videos on the web, but it is unclear whether it works with unsegmented web videos and unsegmented and untrimmed videos, such as egocentric videos [50]. These settings are harder than our experimental setting. Towards industrial use, we need to develop a method that accepts these inputs. In addition, to confirm the generalizability across domains, we also need to extend the method to other domains, such as the biochemical domain [51]. Our approach utilizes a large-scale dataset, which is not always available on the web in other domains. Thus, we need to transfer procedural knowledge learned from cooking tasks to others.

Declarations

Funding and/or Conflicts of interests/Competing interests. 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. All of them are research grants from the Japanese government.
Disclosure of potential conflicts of interest. All authors state that no financial/non-financial support has been received from any organization that may have an interest in this work.

Data Availablity Statement

The datasets generated during and/or analysed during the current study are available in our repository[7]: https://github.com/misogil0116/svpc_pp

[7]Currently, the repository is a private mode. After our manuscript has been accepted, I will release the code and dataset.

Appendix A Details of simulator

In this section, we describe the details of the visual simulator for the reproducibility of the proposed method. Given the encoded vectors of the clip sequence and material list $(\boldsymbol{\mathcal{H}}, \boldsymbol{\mathcal{E}}^0)$, the visual simulator, shown in Figure 3 in the main paper, recurrently 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 $\boldsymbol{\mathcal{E}}^{n-1}$, the visual simulator predicts executed actions and involved materials in (1) the action and (2) material selector and it 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. For clarity, we explain the simulation process of the visual simulator at the n-th step.

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 predefined action embedding $\boldsymbol{\mathcal{F}}$. For example, in Figure 3 in the main paper, 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 action probability for each action in $\boldsymbol{\mathcal{F}}$. Then it outputs the selected action vector $\bar{\boldsymbol{f}}_n$ as a weighted sum of the action embedding $\boldsymbol{\mathcal{F}}$ and action probability $\boldsymbol{w}_p$:

$$\boldsymbol{w}_p = \mathrm{MLP}(\boldsymbol{h}_n) \tag{A1}$$

$$\bar{\boldsymbol{w}}_p = \frac{\boldsymbol{w}_p}{\sum_j \boldsymbol{w}_p^j} \tag{A2}$$

$$\bar{\boldsymbol{f}}_n = \bar{\boldsymbol{w}}_p^T \boldsymbol{\mathcal{F}}, \tag{A3}$$

where $\mathrm{MLP}(\cdot)$ represents two-layer MLPs with the sigmoid function and $\boldsymbol{w}_p \in \mathbb{R}^{\|\boldsymbol{\mathcal{F}}\|}$ is the attention distribution over $\|\boldsymbol{\mathcal{F}}\|$ possible actions[8].

Material selector. Based on the action probability $\boldsymbol{w}_p$ and clip vector $\boldsymbol{h}_n$, the material selector outputs the selected material vector $\bar{\boldsymbol{e}}_n$ by choosing materials involved in the clip from the material list $\boldsymbol{\mathcal{E}}^{n-1}$. For example, in Figure 3 in the main paper, 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.

(1) The clip attention chooses relevant materials from the clip vector $\boldsymbol{h}_n$ and action probability $\boldsymbol{w}_p$:

$$\hat{\boldsymbol{h}}_n = \mathrm{ReLU}(\boldsymbol{W}_1 \boldsymbol{h}_n + \boldsymbol{b}_1) \tag{A4}$$

$$\boldsymbol{d}_m = \sigma((\boldsymbol{e}_m^{n-1})^{\mathsf{T}} \boldsymbol{W}_2 [\hat{\boldsymbol{h}}_n;\ \boldsymbol{w}_p]) \tag{A5}$$

[8] $\boldsymbol{w}_p^j$ represents j-th value of $\boldsymbol{w}_p$; thus, Eq.(6) indicates normalization of $\boldsymbol{w}_p$

where $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ are linear and bilinear mapping, $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are biases, and $\boldsymbol{e}_m^{n-1}$ and $\boldsymbol{d}_m$ represent the m-th material vector and its attention weight.

(2) Recurrent attention selects materials based on information from both the current and previous clips. Using the result of clip attention, it computes a soft selection $\boldsymbol{a}^n$ as a material probability for each material in the material list:

$$\boldsymbol{c} = \mathrm{softmax}(\boldsymbol{W}_3\hat{\boldsymbol{h}}_n + \boldsymbol{b}_3) \tag{A6}$$

$$\boldsymbol{a}_m^n = \boldsymbol{c}_1\boldsymbol{d}_m + \boldsymbol{c}_2\boldsymbol{a}_m^{n-1} + \boldsymbol{c}_3\boldsymbol{0} \tag{A7}$$

where $\boldsymbol{W}_3$ is a linear mapping, $\boldsymbol{c} \in \mathbb{R}^3$ is the choice distribution, $\boldsymbol{a}_m^{n-1}$ is the attention weight of the previous clip for each material, $\boldsymbol{a}_m^n$ is the final distribution for each material, and $\boldsymbol{0}$ is a vector of zeros (providing the option not to select any materials). Finally, using the calculated attention weights, the selected material vector $\bar{\boldsymbol{e}}_n$ is computed as the normalized weighted sum of the selected materials.

$$\boldsymbol{\alpha}_m^n = \frac{\boldsymbol{a}_m^n}{\sum_j \boldsymbol{a}_j^n} \tag{A8}$$

$$\bar{\boldsymbol{e}}_n = \sum_m \boldsymbol{\alpha}_m^n \boldsymbol{e}_m^{n-1}. \tag{A9}$$

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 the selected action and material vectors $(\bar{\boldsymbol{f}}_n, \bar{\boldsymbol{e}}_n)$:

$$\boldsymbol{l}_n = \mathrm{ReLU}(\bar{\boldsymbol{f}}_n\boldsymbol{W}_4\bar{\boldsymbol{e}}_n + \boldsymbol{b}_4), \tag{A10}$$

where $\boldsymbol{W}_4$ is a bilinear mapping.

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 current material vector $\boldsymbol{e}_m^{n-1}$:

$$\hat{\boldsymbol{e}}_m = \boldsymbol{a}_n^m\boldsymbol{l}_n + (1 - \boldsymbol{a}_n^m)\boldsymbol{e}_m^{n-1}. \tag{A11}$$

The new m-th material vector $\hat{\boldsymbol{e}}_m$ is assigned to $\boldsymbol{\mathcal{E}}_m^n$, which is forwarded to the next $(n+1)$-th step.

Appendix B Detailed Annotation Process

We additionally annotated ingredients with the rest of the recipes (126 recipes) to the YouCook2-ingredient dataset, and built the YouCook2-ingredient+ dataset.

We increased the dataset size by obtaining the missing videos through YouCook2 author and annotating ingredients with these additional videos by

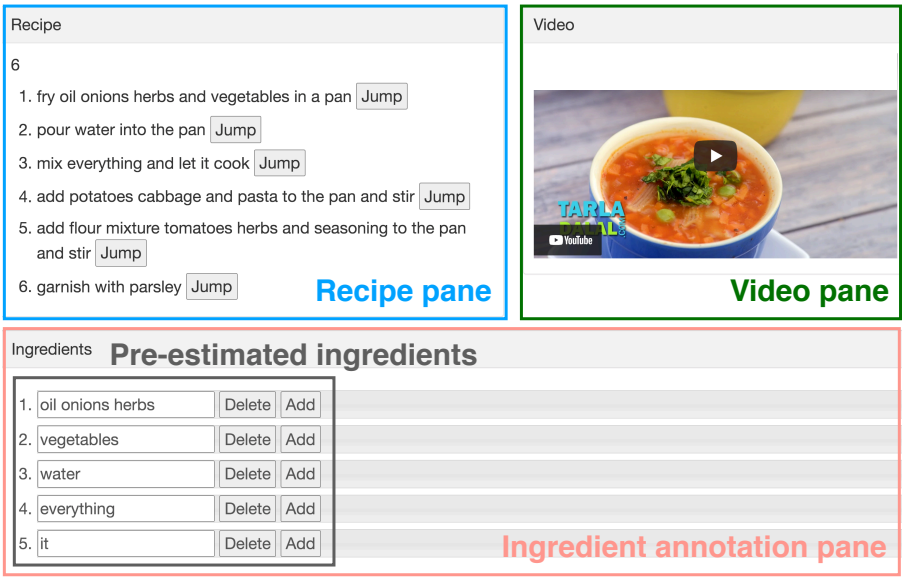

Fig. B1 A screen of our browser-based web annotation tool. Annotators write ingredients that appear in the recipes. To ease annotation, we preliminarily estimated ingredients using the NER method [52] pre-trained on the English recipe flow graph corpus [53], and we set the default values of inputs.

hiring one annotator to use the web tool shown in Fig. B1. This annotation tool presents a recipe, corresponding video, and text boxes for writing ingredients. In this paper, ingredients are defined as raw materials that are necessary to complete the dish. For example, "tomato" and "cucumber" should be written as ingredients, although "salad" should not be written because it represents a mixture of ingredients.

To annotate ingredients easily, "jump" buttons, which allow annotators to see a clip corresponding to a step, are implemented based on the start/end timestamp from the original YouCook2 dataset. Moreover, to encourage annotators to write ingredients easily, this tool displays estimated ingredients using the named entity recognition (NER) model, flair[9] [52] pre-trained on the English recipe flow graph corpus (E-rFG corpus) [53][10]. If the estimated words are not appropriate for the ingredients, the ingredients can be deleted or rewritten.

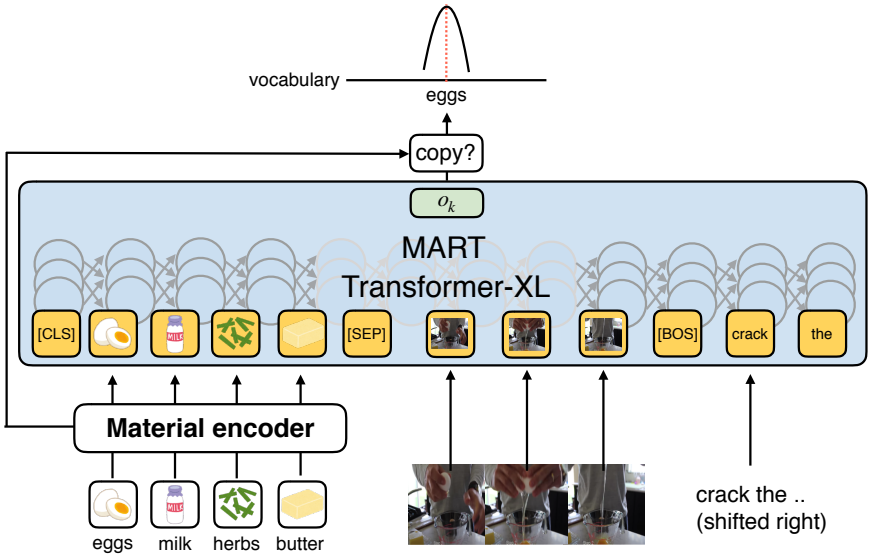

Fig. C2 An overview of our baseline +ingredients (-I) implementation. These models also incorporate the material encoder described in Section 3.2 and copy mechanism into the baselines.

Appendix C Baseline implementation details

As our comparative models, we employed two state-of-the-art transformer-based video captioning models: Transformer-XL [40] and MART [13]. These models originally have no ingredient set in their inputs and copy mechanism in their decoder; thus for a fair comparison, we prepare for additional baseline +ingredient (-I) models, which incorporate the material encoder (Section 3.2) and the copy mechanism into the baselines.

These models are based on the transformer that encodes sequential inputs and decodes a sentence by attending all of the elements in the input sequence. Thus, to fit this characteristic, we concatenate the encoded ingredient and video vectors, and input them to the model, as shown in Fig. C2. When decoding, based on the output of the decoder $\boldsymbol{o}_k$ and ingredient vectors $\boldsymbol{\mathcal{E}}_0$, the copy mechanism calculates the copying gate to make a soft choice between selecting an ingredient from the ingredient set or generating a word from the vocabulary.

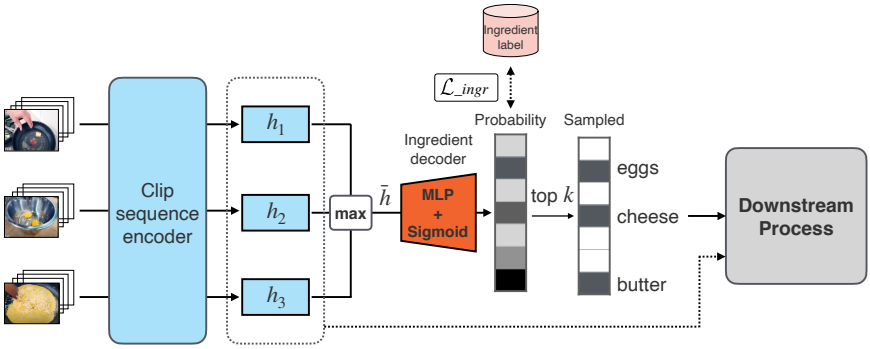

Fig. D3 An overview of how to integrate ingredient decoder into the model.

Appendix D Implementation and Training Details on full prediction settings

Here, we discuss the implementation and training details of the full prediction settings, where the material set is not given, but is predicted from the video clips in advance. To address this, as described in Section 4.7, we added an ingredient decoder of the multi-label classifier and trained the entire model as multi-task learning.

Fig. D3 shows how to integrate the ingredient decoder into the model. The ingredient decoder consists of a two-layered MLP with a sigmoid function, and converts $\hat{\boldsymbol{h}}$ of a max-pooled vector of clip vectors into a probability vector $\in \mathbb{R}^q$ of materials, where q indicates the number of unique ingredients appearing more than three times in the training set (we obtained $q = 668$ in the experiment). During training, we compute the ingredient decoder loss $\mathcal{L}_{ingr}$, which is an asymmetric loss [36] on the multi-label classification settings, and add it to the total loss defined in Eq (4). Note that we adopt teacher-forcing [54] to stabilize the training; while the models learn using the ground-truth ingredients for the downstream process in the training phase, they generate a recipe based on predicted ingredients at the inference phase (we sample the top $k = 15$ ingredients from the probability). Another modification of the model is to remove the copy mechanism because we find that it degrades the captioning performance with this setting.

References

[1] Zhou, L., Xu, C., Corso, J.J.: Towards automatic learning of procedures

[9] https://github.com/flairNLP/flair

[10] In the tag definitions of the E-rFG corpus, we display food entities as the estimated ingredients. These entities cannot be directly used for our dataset because the definition of food slightly differs from the ingredient definition in this paper (for example, "it" or "salad" are recognized as food in the E-rFG corpus). Therefore, we asked annotators to delete or rewrite ingredients if they were not appropriate.

from web instructional videos. In: Proc. AAAI, pp. 7590–7598 (2018)

[2] Alayrac, J.-B., Bojanowski, P., Agrawal, N., Sivic, J., Laptev, I., Lacoste-Julien, S.: Unsupervised learning from narrated instruction videos. In: Proc. CVPR, pp. 4575–4583 (2016)

[3] Alayrac, J.-B., Sivic, J., Laptev, I., Lacoste-Julien, S.: Joint discovery of object states and manipulation actions. In: Proc. ICCV, pp. 2127–2136 (2017)

[4] Miech, A., Zhukov, D., Alayrac, J.-B., Tapaswi, M., Laptev, I., Sivic, J.: HowTo100M: learning a text-video embedding by watching hundred million narrated video clips. In: Proc. ICCV, pp. 2630–2640 (2019)

[5] Miech, A., Alayrac, J.-B., Smaira, L., Laptev, I., Sivic, J., Zisserman, A.: End-to-end learning of visual representations from uncurated instructional videos. In: Proc. CVPR, pp. 9879–9889 (2020)

[6] Shi, B., Ji, L., Liang, Y., Duan, N., Chen, P., Niu, Z., Zhou, M.: Dense procedure captioning in narrated instructional videos. In: Proc. ACL, pp. 6382–6391 (2019)

[7] Shi, B., Ji, L., Niu, Z., Duan, N., Zhou, M., Chen, X.: Learning semantic concepts and temporal alignment for narrated video procedural captioning. In: Proc. ACMMM, pp. 4355–4363 (2020)

[8] Escorcia, V., Heilbron, F.C., Niebles, J.C., Ghanem, B.: DAPs: deep action proposals for action understanding. In: Proc. ECCV, pp. 768–784 (2016)

[9] Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: towards real-time object detection with region proposal networks. In: Proc. NeurIPS, pp. 91–99 (2015)

[10] Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., Zagoruyko, S.: End-to-end object detection with transformers. In: Proc. ECCV, pp. 213–229 (2020)

[11] Donahue, J., Hendricks, L.A., Rohrbach, M., Venugopalan, S., Guadarrama, S., Saenko, K., Darrell, T.: Long-term recurrent convolutional networks for visual recognition and description. In: Proc. CVPR, pp. 2625–2634 (2015)

[12] Tan, G., Liu, D., Wang, M., Zha, Z.-J.: Learning to discretely compose reasoning module networks for video captioning. In: Proc. IJCAI, pp. 745–752 (2020)

[13] Lei, J., Wang, L., Shen, Y., Yu, D., Berg, T., Bansal, M.: Mart: memory-augmented recurrent transformer for coherent video paragraph captioning. In: Proc. ACL, pp. 2603–2614 (2020)

[14] Xiong, Y., Dai, B., Lin, D.: Move forward and tell: a progressive generator of video descriptions. In: Proc. ECCV, pp. 489–505 (2018)

[15] Zhou, L., Kalantidis, Y., Chen, X., Corso, J.J., Rohrbach, M.: Grounded video description. In: Proc. CVPR, pp. 6578–6587 (2019)

[16] Bosselut, A., Levy, O., Holtzman, A., Ennis, C., Fox, D., Choi, Y.: Simulating action dynamics with neural process networks. In: Proc. ICLR (2018)

[17] Amac, M.S., Yagcioglu, S., Erdem, A., Erdem, E.: Procedural reasoning networks for understanding multimodal procedures. In: Proc. CoNLL, pp. 441–451 (2019)

[18] Nishimura, T., Hashimoto, A., Ushiku, Y., Kameko, H., Mori, S.: State-aware video procedural captioning. In: Proc. ACMMM (2021)

[19] Park, J.S., Rohrbach, M., Darrell, T., Rohrbach, A.: Adversarial inference for multi-sentence video description. In: Proc. CVPR, pp. 6598–6608 (2019)

[20] Zhou, L., Zhou, Y., Corso, J.J., Socher, R., Xiong, C.: End-to-end dense video captioning with masked transformer. In: Proc. CVPR, pp. 8739–8748 (2018)

[21] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: Proc. NeurIPS, pp. 5998–6008 (2017)

[22] Kiddon, C., Ponnuraj, G.T., Zettlemoyer, L., Choi, Y.: Mise en Place: unsupervised interpretation of instructional recipes. In: Proc. EMNLP, pp. 982–992 (2015)

[23] Maeta, H., Sasada, T., Mori, S.: A framework for procedural text understanding. In: Proc. IWPT, pp. 50–60 (2015)

[24] Jermsurawong, J., Habash, N.: Predicting the structure of cooking recipes. In: Proc. EMNLP, pp. 781–786 (2015)

[25] Pan, L., Chen, J., Wu, J., Liu, S., Ngo, C.-W., Kan, M.-Y., Jiang, Y.-G., Chua, T.-S.: Multi-modal cooking workflow construction for food recipes. In: Proc. ACMMM, pp. 1132–1141 (2020)

[26] Nishimura, T., Hashimoto, A., Ushiku, Y., Kameko, H., Yamakata, Y.,

Mori, S.: Structure-aware procedural text generation from an image sequence. IEEE Access **9**, 2125–2141 (2020)

[27] Gupta, A., Durrett, G.: Tracking discrete and continuous entity state for process understanding. In: Proc. NAACL Workshop SPNLP, pp. 7–12 (2019)

[28] Santoro, A., Faulkner, R., Raposo, D., Rae, J., Chrzanowski, M., Weber, T., Wierstra, D., Vinyals, O., Pascanu, R., Lillicrap, T.: Relational recurrent neural networks. In: Proc. NeurIPS, pp. 7299–7310 (2019)

[29] Dalvi, B., Huang, L., Tandon, N., Yih, W.-t., Clark, P.: Tracking state changes in procedural text: a challenge dataset and models for process paragraph comprehension. In: Proc. NAACL, pp. 1595–1604 (2018)

[30] Pennington, J., Socher, R., Manning, C.: Glove: global vectors for word representation. In: Proc. EMNLP, pp. 1532–1543 (2014)

[31] Devlin, J., Chang, M.-W., Lee, K., Toutanova, K.: Bert: pre-training of deep bidirectional transformers for language understanding. In: Proc. NAACL, pp. 4171–4186 (2019)

[32] Sun, C., Myers, A., Vondrick, C., Murphy, K., Schmid, C.: Videobert: a joint model for video and language representation learning. In: Proc. ICCV, pp. 7464–7473 (2019)

[33] See, A., Liu, P.J., Manning, C.D.: Get to the point: summarization with pointer-generator networks. In: Proc. ACL, pp. 1073–1083 (2017)

[34] Jang, E., Gu, S., Poole, B.: Categorical reparametrization with gumble-softmax. In: Proc. ICLR (2017)

[35] Mintz, M., Bills, S., Snow, R., Jurafsky, D.: Distant supervision for relation extraction without labeled data. In: Proc. ACL-IJCNLP, pp. 1003–1011 (2009)

[36] Zamir, N., Noy, A., Friedman, I., Protter, M., Zelnik-Manor, L.: Asymmetric loss for multi-label classification. arXiv (2020)

[37] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proc. CVPR, pp. 770–778 (2016)

[38] Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: Proc. ICML, pp. 448–456 (2015)

[39] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Proc. ICLR, USA

[40] Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q., Salakhutdinov, R.: Transformer-xl: attentive language models beyond a fixed-length context. In: Proc. ACL, pp. 2978–2988 (2019)

[41] Papineni, K., Roukos, S., Ward, T., Zhu, W.-J.: BLEU: a method for automatic evaluation of machine translation. In: Proc. ACL, pp. 311–318 (2002)

[42] Lin, C.-Y., Och, F.J.: Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics. In: Proc. ACL, pp. 605–612 (2004)

[43] Banerjee, S., Lavie, A.: METEOR: an automatic metric for MT evaluation with improved correlation with human judgments. In: Proc. ACL Workshop IEEMMTS, pp. 65–72 (2005)

[44] Vedantam, R., Zitnick, C.L., Parikh, D.: CIDEr: consensus-based image description evaluation. In: Proc. CVPR, pp. 4566–4575 (2015)

[45] Chen, J., Ngo, C.-w.: Deep-based ingredient recognition for cooking recipe retrieval. In: Proc. ACMMM, pp. 32–41 (2016)

[46] van der Maaten, L., Hinton, G.: Visualizing data using t-sne. Journal of Machine Learning Research **9**, 2579–2605 (2008)

[47] Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: NeurIPS, pp. 3111–3119 (2013)

[48] Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv (2015)

[49] Salvador, A., Hynes, N., Aytar, Y., Marin, J., Ofli, F., Weber, I., Torralba, A.: Learning cross-modal embeddings for cooking recipes and food images. In: Proc. CVPR, pp. 3020–3028 (2017)

[50] Damen, D., Doughty, H., Farinella, G.M., Fidler, S., Furnari, A., Kazakos, E., Moltisanti, D., Munro, J., Perrett, T., Price, W., Wray, M.: Scaling egocentric vision: The EPIC-KITCHENS dataset. In: Proc. ECCV, pp. 720–736 (2018)

[51] Nishimura, T., Sakoda, K., Hashimoto, A., Ushiku, Y., Tanaka, N., Ono, F., Kameko, H., Mori, S.: Egocentric biochemical video-and-language dataset. In: Proc. CLVL, pp. 3129–3133 (2021)

[52] Akbik, A., Blythe, D., Vollgraf, R.: Contextual string embeddings for sequence labeling. In: Proc. COLING, pp. 1638–1649 (2018)

[53] Yamakata, Y., Mori, S., Carroll, J.: English recipe flow graph corpus. In: Proc. LREC, pp. 5187–5194 (2020)

[54] Williams, R.J., Zipser, D.: A learning algorithm for continually running fully recurrent neural networks. Neural Computation **1**, 270–280 (1989)