Neural Machine Translation for Conditional Generation of Novel Procedures

Joppe Geluykens
IBM Research Europe, Zürich
jge@zurich.ibm.com

Sandra Mitrović
IDSIA, Switzerland
sandra.mitrovic@idsia.ch

Carlos Ortega Vázquez
KU Leuven
carloseduardo.ortegavazquez@kuleuven.be

Teodoro Laino
IBM Research Europe, Zürich
teo@zurich.ibm.com

Alain Vaucher
IBM Research Europe, Zürich
ava@zurich.ibm.com

Jochen De Weerdt
KU Leuven
jochen.deweerdt@kuleuven.be

Abstract

Procedural knowledge is generally dispersed across many experts within or across organizations which might lead to inefficiencies and redundancy. Historically, computers have been well suited to store procedural knowledge but they have lacked the capability to produce natural language text. Nonetheless, recent advances in machine learning permit a higher linguistic coherence which benefits applications with longer text outputs such as procedures. This work closes the gap between human experts and computers by proposing a framework for automatic, computer generation of procedures based on neural machine translation and the BART model. Furthermore, we define two benchmark problems for procedure generation and establish a set of evaluation metrics that can be used as a reference in further work. We demonstrate the potential of this solution on the task of generating cooking recipes based on available ingredients. The evaluation results on the Recipe1M dataset showcase the method's superiority over other, fairly novel, neural architectures.

1. Introduction

New advances in Natural Language Generation (NLG) have led to improvements for applications such as machine translation [1], text simplification [2] or summarization [3], and data-to-text generation [4]. While a growing number of works have proposed models for NLG, we observe very few works focusing on procedure generation [5, 6, 7]: a procedure can be represented as a long sequence of text in which a set of tasks is described in an orderly fashion.

Procedure generation could benefit a diverse range of domains that require precise set of instructions in order to produce a certain target or achieve certain state, e.g. molecule synthesis for drug discovery, meal preparation using specific ingredients [7], furniture/appliances assembling, enforcing take-off/landing checklists in aviation, generation of test use cases for software development [8]. These typically encompass a number of requirements to be satisfied, a particular order of tasks to be respected and, in some cases, depending on circumstances, the adaptation of the granularity level of the conveyed tasks. In the case of aviation, a proper checklist must be complete and specific while preserving the order of tasks [9]. Manual task-lists in aviation are prone to omission since they are not standardized; the order is most relevant for starting engines and its related systems. Many unstandardized checklists suffer from ambiguity by using terms such as ”check and set” instead of ”altimeter at 30.10”.

Table 1: Example of a generated procedure after applying the proposed method on cooking recipes.

(a) Input requirements (ingredients)	(b) Generated procedure (title and instructions)
Requirements • 2 tablespoons ground coffee (optional) • 5-6 yellow onion skins, only • 8 eggs • water • 1 tablespoon oil	**Target product** Eggs in Onion Skin **Tasks** 1. Beat the eggs in a bowl. 2. Add the onion skins and coffee (if using). 3. Add enough water to cover the onion skins. 4. Heat the oil in a frying pan. 5. Fry the eggs in the oil until they are cooked through. 6. Serve with the onion skins.

URI: https://hdl.handle.net/10125/70744
978-0-9981331-4-0

Relying on generated procedures can lower the cost of acquiring procedural knowledge and allow more time for creative tasks [4]. Another benefit is the centralization of otherwise dispersed procedural knowledge among the task-specific experts: the knowledge of synthesizing a molecule can be shared to others even if the human expert is not available.

One of the most prominent examples of procedure generation comes from the field of computational cooking [10] and recipe generation [7]. An example of a procedure generated by an NLG model trained on cooking recipes is shown in Table 1. A recipe contains at least a set of ingredients and instructions in which the order of tasks is essential. Furthermore, a recipe can be arbitrarily long so preserving global relational coherence (i.e. task-based meaningfulness) becomes a challenge [4, 11]. For example, a correctly generated procedure for preparing bread should first suggest making the dough before putting it into the oven.

Previous works on recipe generation have mostly relied on well-established methods such as directed acyclic graph (DAG) [6] or recurrent neural network (RNN) [5]. Despite the state-of-the-art performance for prediction tasks for long text sequences (e.g. translation) [12], with the exception of [7], no work has addressed procedure generation using the Transformer [13] as the model architecture. In [7], a renowned Generative Pretrained Transformer (GPT) architecture is used to develop RecipeGPT. In this work, we propose to go a step further by adding a bidirectional encoding of the input, motivated by the fact that requirements (ingredients) for a procedure correspond to an unordered list so the embeddings of requirements should not be order-dependent. We investigate BART [14], a state-of-the-art model that is based on bidirectional auto-regressive Transformers and combines the benefits of both the bidirectional encoder model and GPT. Specifically, we consider the study of recipe generation under the neural machine translation framework: ingredients, as predefined requirements, are translated into a sequence of specific tasks describing how to make the target product. We demonstrate the benefits of exploiting BART, not only as compared to other machine translation methods based on LSTM, CNNs or the Transformer model, but based on the GPT-based RecipeGPT [7] as well. The overview of the proposed method is given in Figure 1. Despite the focus on the recipe generation, our work can be extended to other procedure-based applications.

Our contribution to the current literature is four-fold:

- Formalising the problem of the procedure generation,

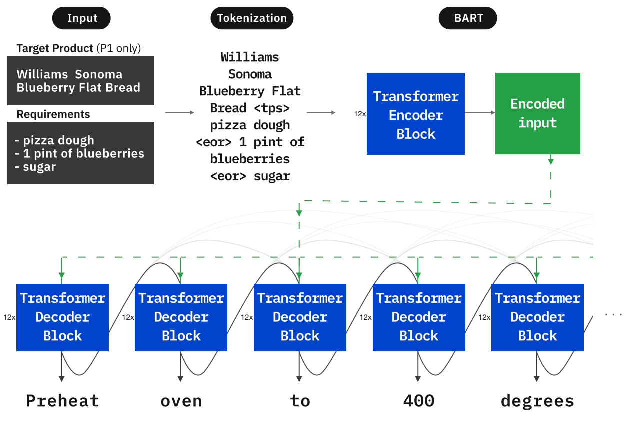
Input
Tokenization
BART
Target Product (P1 only)
Williams Sonoma Blueberry Flat Bread
Requirements
- pizza dough
- 1 pint of blueberries
- sugar
Williams Sonoma Blueberry Flat Bread <tps> pizza dough <eor> 1 pint of blueberries <eor> sugar
12x
Transformer Encoder Block
Encoded input
Transformer Decoder Block
Preheat
oven
to
400
degrees

Figure 1: Overview of the system. The input requirements and target product (if $P1$) are first tokenized; the target product is followed by a separator token (<tps>), the end of a requirement (EOR) is indicated using the <eor> token. Consequently, the input tokens are embedded by the encoder. The encoded input is fed to the decoder at each step, and generates output tokens autoregressively.

- Addressing procedure generation task by means of the state-of-the-art NLG / neural machine translation method BART, while ensuring requirement coverage, as well as the proper task order and granularity in the generated procedures,

- Defining and implementing metrics to evaluate model performance on the problem of procedure generation,

- Implementing and benchmarking four machine translation methods for generating procedures within the proposed methodology.

In the remainder of the paper, Section 2 briefly discusses related work, Section 3 provides the formalisation of the problem at hand, Section 4 explains the details of our methodology, Section 5 details the experimental setup while Section 6 presents the experimental results and finally, Section 7 concludes the paper providing directions for future research.

2. Related Work

Literature on NLG focuses on either data-to-text (i.e. multi-modal), or text-to-text (i.e. uni-modal) generation [4]. Within the computational cooking field, the task of recipe generation is fairly commonly studied; we can find both of these in recipe generation. On the one hand, studies on data-to-text generation [15, 16] attempt to produce recipes from images and ingredients, but do not generalize beyond the recipe application. On the other hand, early studies on text-to-text generation [17, 18, 6] have proposed techniques that generate a natural language text given a representation of the recipe. More recent studies [5, 19, 7] have used neural network architectures. For example, in [5] RNNs have been exploited for recipe generation, while in [19] the proposed solution exploits the attention mechanism.

Probably the closest work to ours is the very recent work reported in [7], which is based on the GPT model. However, that study is more oriented toward providing a particular tool for generating recipes from ingredients and reversely, generating ingredients from recipes (while in both cases providing the target).

It is important to emphasise that unlike our work, none of the mentioned studies address the aspects of task order, granularity and requirement coverage. Thus in this study, we are paying special attention to closing these gaps. By utilising the mechanism behind the BART model, we aim at (and demonstrate the feasibility of) addressing this aspect in a very effective way.

3. Problem statement

First, we define the concepts required to discuss about the context of procedure generation. A vocabulary for representing linked data related to procedures and their execution has been introduced in [20, 21]. Inspired by this vocabulary, we expand and formalise the concept of procedures and generation thereof.

Let us denote by R a set of input requirements and by T a set of tasks (instructions). A **procedure** Pr is a tuple $\langle tp, M_{Pr}\rangle$, containing a textual description of the target **product** tp and a set of methods $M_{Pr} \subset M\colon R \times T$ for obtaining the product tp. A **method** $m \in M_{Pr}$ is a tuple $\langle R_m, T_m\rangle$ containing a set of requirements $R_m \subset R$ and a set of **tasks** $T_m = \{t : t \text{ is a sentence describing a task}\} \subset T$ that need to be performed in order to obtain the product tp. One procedure Pr can have several alternative methods $m \in M$ to obtain the same product tp. A **requirement** $r \in R$ is a tuple $\langle o, q, \{true, null\}\rangle$ containing the textual representation of an object o (*what* is required), a numerical or textual quantity q (*how much* of the object o is required) and a boolean set to $true$ when this requirement is non-essential (optional) to complete the method, otherwise it is $null$. Requirements for a method m are necessary conditions for being able to execute the tasks for m.

Following [21], we define an ordering of the tasks, to describe a dependency relation where one task $t_A \in T$ relies on the completion of one or more prior tasks $t_i \in T, i \neq A$. Formally, we use the dependency relation $\prec_m\colon T_m \times T_m$ where $\prec_m$ defines a strict partial order over the set of tasks T_m for a method m [21]. The requirements are treated as not having any internal order, i.e. they could be used in any order in the sequence of tasks, independent of the order in which they are listed.

Using the terminology above, we formulate the two problems that we consider in the scope of this work. Both problems are related to the general situation where a system or agent is operating in an environment for which the conditions are known, and it is of interest to know what the possible procedures are for operating in this environment. In such a scenario, the requirements R_m are the conditions of the operating environment and constant for all methods $m \in M_{Pr}$. When also the product tp is known, all parts of a procedure Pr are defined, except the set of tasks $T_m, \forall m \in M_{Pr}$.

Problem 1: Generation of tasks given requirements and target product $P1\colon R \times TP \longrightarrow T$. Problem 1 is concerned with generating a set of tasks T_m for a method $m \in M_{Pr}$, when requirements R_m and product tp are known. The problem is defined as sampling the conditional probability of a set of tasks T_m. Each task $t_m^i \in T_m$ is conditioned on the requirements R_m, the product tp and all previously sampled tasks $t_m^1, ..., t_m^{i-1}$:

$$\begin{aligned} t_m^i \sim\ & P(T_m^i \mid R = R_m,\ TP = tp, \\ & \qquad T_m^1 = t_m^1, ..., T_m^{i-1} = t_m^{i-1}) \\ \text{for } i = & 1, ..., |T_m|,\ m \in M_{Pr}. \end{aligned} \tag{1}$$

Problem 2: Generation of target product and corresponding tasks given requirements $P2\colon R \longrightarrow TP \times T$. Problem 2 makes no assumption of the product tp. Instead, product tp is sampled from the probability distribution of target products TP conditioned on the requirements R_m. The tasks are then sampled from the same distribution as in Problem 1, conditioned on the sampled product tp instead of a given tp:

$$\begin{aligned} tp \sim\ & P(TP \mid R = R_m), \\ t_m^i \sim\ & P(T_m^i \mid R = R_m,\ TP = tp, \\ & \qquad T_m^1 = t_m^1, ..., T_m^{i-1} = t_m^{i-1}) \\ \text{for } i = & 1, ..., |T_m|,\ m \in M_{Pr}. \end{aligned} \tag{2}$$

4. Methodology

Once we have defined the problem scope, we first elaborate on the model architecture used for procedure generation. Then, we describe the metrics we propose for evaluating procedures generated by the model.

4.1. Model architecture

Although other techniques might be feasible for procedure generation, we opt for the use of neural machine translation models because of their general applicability to natural language of any kind. For the purpose of this research, we assess the capability of BART [14], a state-of-the-art neural machine translation modeling technique, for solving problems $P1$ and $P2$.

All models considered in this work use the encoder-decoder architecture for sequence-to-sequence learning with neural networks, as first described in [22]. The general idea of the encoder-decoder neural network architecture is to embed (encode) the whole input sequence to a hidden state of predefined dimensions and consequently construct (decode) the output sequence from that same hidden state [22]. In the case of machine translation, both input and output sequences consist of tokenized sentences. Every token is one entry in the sequence. For consistency with [22], we note the input sequence of tokens as $(x_1, ..., x_T)$, with T the length of the input sequence. Similarly, the output sequence of tokens is noted as $(y_1, ..., y_T')$, with T' the length of the output sequence. For the translation task, every output token is conditioned on all previous input tokens. This corresponds to sampling the conditional distributions as described in Section 3.

Bidirectional and auto-regressive attention-based encoder-decoder. The bidirectional and auto-regressive Transformer model (BART) builds on the standard Transformer architecture, but uses other advances in the literature that refer to optimizing the pretraining of Transformers [14]. BART combines a 12-layer bidirectional encoder (as in [12]) with a 12-layer auto-regressive decoder. BART uses a number of self-supervised pretraining tasks where the goal is to reconstruct the original input. The best performance is achieved using the following input transformations:

- Random shuffling of sentence order.
- In-filling of pieces of arbitrary length which are masked.

The idea is that the BART model is pretrained with these tasks and finetuned on the downstream task afterwards. In the context of procedure generation, we use a BART model trained to perform the above pretraining tasks on the CNN-DailyMail summarization dataset. We then switch to the translation task to finetune the model on our training data to predict procedures. We notice that finetuned BART performs better on all evaluation metrics and converges faster compared to training BART from scratch on the procedure generation data.

The intuition for using BART in the context of procedure generation is that the order of the requirements is not relevant for predicting tasks (and target product). It is up to the model to, at generation time, come up with a correct order in which to use the requirements. The auto-regressive decoder can exploit the bidirectional encoding of the entire set of input requirements (and target product) to predict the next task. Furthermore, a coherent task order is encouraged through the auto-regressive decoder, since the prediction of a task is conditioned on all previously predicted tasks.

4.2. Evaluation

To the best of our knowledge, no standard set of evaluation metrics exists for the problems described in Section 3. Care should be taken in selecting the metrics suitable for evaluating the results. In this section, all metrics that we deem appropriate for $P1$ and $P2$ are formulated at three levels: word-level (comparing words), task-level or product-level (comparing single tasks or a single product) and task set-level (comparing whole set of tasks).

Word-level

Perplexity. The perplexity value $PP = 2^{NLL}$, with NLL being the negative log-likelihood loss, is an indication of the number of possible tokens that the model deems probable for predicting the next token. A lower value corresponds to the model assigning the full probability mass to only a couple of possible tokens, making it more sure about its prediction [23].

Task- or product-level

BLEU score. The BLEU score attributes a higher value to translations that have more words in common with reference translations [24]; it measures precision, that is, the relevance of the generated text regarding the reference or instructions. We use the ground truth as reference to compute the BLEU score.

ROUGE score. In contrast to BLUE, ROUGE measures recall since it compares the predicted text and the recipe reference [25].

Embedding distance. A semantically more meaningful way to automatically score translations against the ground truth, is by comparing the distance between them in the embedding space of a semantically aware language model. We choose the recently proposed BERTScore [26] for this purpose, which compares each token in the ground truth sentence x with the corresponding token in the generated sentence $\hat{x}$. First, a contextual embedding vector is extracted from a trained BERT model [12] for all ground truth tokens and generated tokens. Then, the pairwise cosine similarity between two embedding vectors $\mathbf{x}$ and $\hat{\mathbf{x}}$ is used to compute recall and precision as follows: $R_{BERT} = \frac{1}{|x|} \sum_{x_i \in x} \max_{\hat{x}_j \in \hat{x}} \mathbf{x_i}^\top \hat{\mathbf{x}}_\mathbf{j}$, $P_{BERT} = \frac{1}{|\hat{x}|} \sum_{\hat{x}_j \in \hat{x}} \max_{x_i \in x} \mathbf{x_i}^\top \hat{\mathbf{x}}_\mathbf{j}$. The final BERTScore we report, corresponds to the F1-measure: $F_{BERT} = 2 \frac{P_{BERT} \cdot R_{BERT}}{P_{BERT} + R_{BERT}}$.

Task set-level
Task order. We aim to evaluate how well the order between tasks is preserved in generated procedures (e.g.: peel avocado $\prec_m$ mash avocado, mash avocado $\not\prec_m$ peel avocado.) To this end, we opt for computing the Kendall τ correlation metric to rank the predicted task order against the ground truth task order [27]. It can be computed as follows: $\frac{n_c - n_d}{n(n-1)/2}$, where n_c and n_d represent the concordant and discordant pairs while n is the number of observations.
Requirement coverage. The requirement coverage is described as a percentage of input requirements that are present in the predicted tasks: $\frac{|\text{ requirements used in generated tasks}|}{|\text{ requirements in the input}|} * 100$. This metric is called requirement coverage, in accordance with the terminology used in [28].

Essential requirement coverage. This metric is a specific case of the Requirement coverage metric, but the percentage is computed only taking essential requirements into account.

5. Experimental Setup

For validating our hypothesis stating that the Transformer architecture can be used to generate task descriptions containing a sequence of actions, we use the domain of cooking recipes. This section describes the experimental setup, including implementation details, for the specific recipe generation case study. The complete source code will be released upon acceptance.

5.1. Data

We perform our experiments on the Recipe1M dataset [29], which compiled a multi-modal collection of over one million free-text recipes and corresponding images scraped from the web. For our use case, we only use the free-text recipes. Each recipe (procedure) consists of ingredients (requirements), instructions (tasks) and a title (target product). For the purpose of consistency and reproducibility, we use the same train, validation and test splits as provided by [29]. Table 2 describes the various statistics across the different splits.

Table 2: Basic statistics of the Recipe1M dataset.

	Train	Validation	Test
Number of recipes	720639	155036	154045
Mean instruction length (tokens)	126.52	126.80	125.92
Mean number of ingredients	9.33	9.33	9.33

Table 3b shows the ground truth example from the Recipe1M dataset for the example shown in the beginning of this paper (see Table 1).

5.2. Data preparation

A Python framework, *proc-gen*, tailored to procedure generation was implemented to support any kind of procedure through the `Procedure` class. The types used to represent the training data are general enough to use for tasks other than cooking recipe generation. Furthermore, although here we handle only the Recipe1M dataset, the *proc-gen* framework allows for loading multiple datasets into `Procedures`.

The text encoding is performed as follows. First, both source and target sentences are transformed by splitting words on punctuation using the Moses tokenizer [30]. This tokenizer is used, so that in the next step we can make use of open-source vocabularies learned from English text [31]. Byte-pair encoding (BPE), as first described in the context of machine translation by [32], is then used to encode all word units to an integer index. This mapping is stored in a dictionary for easy encoding and decoding.

5.3. Model training

All baselines and model implementations are implemented in PyTorch [33] using the *fairseq* [31] framework by Facebook AI Research. Fairseq provides reference implementations for the models of interest.

All models are trained with distributed training, using data parallelism. Training a single model (BART) on 20 GPUs takes about 1h15m per epoch.

5.4. Baselines

For each of the two problems defined in Section 3, we train three baseline models as follows.

1. Long short-term memory (LSTM). The LSTM [34] is a modified version of a recurrent neural network (RNN) [35], that is better suited for learning long-range dependencies by keeping an extra internal state. This state stores important information to be used later in the sequence. In the context of procedure generation, the model could learn to store information about the input requirements and/or target product in the internal state.

2. Convolutional encoder-decoder. Another approach to modeling the encoder-decoder architecture replaces RNNs with convolutional neural networks (CNN) for both the encoder and decoder. In [36], attention is also applied to allow each decoder layer to focus on specific input tokens when generating the next output token. The decoder computes an attention score per input token based on the current hidden state and the ground truth output token from the previous time step. In the case of procedure generation, this allows the decoder to focus on a specific requirement and/or the target product while generating a certain output task.

3. Attention-based encoder-decoder. Instead of RNNs and CNNs, Vaswani et al. [13] rely only on (self-)attention layers to construct an encoder-decoder architecture, which they call Transformer. The Transformer uses three types of attention [13]: the encoder-decoder attention, self-attention in the encoder and self-attention in the decoder. In the Transformer model, one *Transformer encoder block* consists of an attention layer, followed by a feedforward neural network layer. For the *Transformer decoder block*, the first layer is the masked attention layer, followed by the encoder-decoder attention layer and finally a feedforward neural network layer. For both encoder and decoder blocks, a residual connection is added to each layer output. The original Transformer architecture by [13] uses 6 encoder blocks and 6 decoder blocks. To incorporate positional awareness, positional encoding is added to the input sequence.

4. Generative Pretrained Transformer2 (GPT2). The GPT2, is a direct scale-up of GPT with more than 10x the parameters, and as such, yet another Transformer-based model. GPT2 is using only Transformer decoder blocks and unlike BART it is not bidirectional. GPT2 is an "auto-regressive" model meaning that the output (token) of the previous step is added to the sequence of inputs in the next step.

Table 3: An example of recipe with optional input requirements. Figure (a) displays a set of input requirements (with coffee as optional requirement), (b) ground truth recipe, (c) recipe generated for $P2$ by the by the BART model. The BART model accurately interpreted the coffee requirement as optional and incorporated that information in the generated tasks (notice the underlined part "if using").

(a) Input requirements (ingredients)	
Requirements	
• 2 tablespoons ground coffee (optional)	• 8 eggs
• 5-6 yellow onion skins, only	• water
	• 1 tablespoon oil
(b) Ground truth procedure (verbatim from the dataset)	(c) Generated procedure (produced by BART model)
Target product Egyptian Slow Cooked Eggs (Beid Hamine)	**Target product** Eggs in Onion Skin
Tasks	**Tasks**
1. Add all ingredients to a pot and add water to cover (about two inches above products in pot) and bring to simmer over the lowest heat possible.	1. Beat the eggs in a bowl.
2. Simmer for 6-8 hours or overnight.	2. Add the onion skins and coffee (if using).
3. Peel and slice.	3. Add enough water to cover the onion skins.
4. Serve with ful maddamas or as a garnish for stews.	4. Heat the oil in a frying pan.
	5. Fry the eggs in the oil until they are cooked through.
	6. Serve with the onion skins.

5.5. Evaluation metrics

For standard evaluation metrics like BLEU, ROUGE and embedding distance (BERTScore [26]), an existing reference implementation is used [37]. Custom evaluation metrics are implemented as follows:

Task-order We implement the Kendall correlation τ by using a 1-based index of the task in the recipe. The rank of each predicted task is equal to the rank of the corresponding task in the ground truth. We assume that each predicted task has such a corresponding task in the ground truth, and we match predicted tasks with ground truth tasks based on their distance in sentence embedding space, using the BERTScore [26]. If the BERTScore of a (ground truth task, predicted task) pair is above 0.5, they are considered a match.

Table 4: Results on 4000 test examples for Problem $P1$ (to the left; tackling generation of tasks given both requirements and target product) and Problem $P2$ (to the right; addressing generation of both target product and tasks given requirements only). Best results per measure and problem are shown in boldface.

	$P1$					$P2$				
	LSTM	CNN	Transf.	GPT2	BART	LSTM	CNN	Transf.	GPT2	BART
Word-level										
Perplexity	83.82	11.79	7.59	**4.39**	5.02	22.25	18.77	8.76	4.79	**4.54**
Task-level										
BLEU	2.96	7.07	3.95	**11.02**	9.24	2.01	2.93	6.00	**12.55**	6.16
ROUGE-1	0.22	0.31	0.25	**0.45**	0.41	0.19	0.22	0.30	**0.46**	0.40
METEOR	0.21	0.24	0.27	0.33	**0.34**	0.19	0.23	0.27	**0.35**	0.30
BERTScore	0.36	0.53	0.37	**0.54**	0.49	0.42	0.44	0.46	**0.55**	0.51
Task set-level										
Req. Cov. (%)	23.64	53.60	60.53	**79.03**	67.35	22.70	50.02	53.81	82.77	**91.03**
Essential Req. Cov. (%)	24.11	55.21	62.37	**81.39**	69.68	24.46	52.37	56.49	84.91	**93.54**
Kendall τ	0.13	0.21	0.13	0.34	**0.39**	0.23	0.13	0.20	**0.33**	0.31

Requirement coverage For each requirement in the source, we verify that the words of object in the requirement are present in the generated set of tasks.

6. Results

In this section we provide the quantitative results with respect to the evaluation measures proposed in Section 3, as well as qualitative results related to specific aspects of generated procedures.

Quantitative results We report the average results for considered evaluation measures on a corpus of 4000 randomly selected test examples using different models (LSTM, CNN, Transformer, GPT-2 and BART), for problems $P1$ and $P2$ in Table 4. The results show that Transformer-based architectures (Transformer, GPT-2 and BART) outperform the other baselines on all the metrics for both $P1$ and $P2$. Specifically, for $P1$, GPT-2 outperforms the other methods except BART in METEOR and Kendall τ, which demonstrates BART efficiency regarding task order. On $P2$, BART outperforms its competitors in coverage metrics and perplexity, while GPT-2 is the best on the task-level metrics.

Requirement coverage Judging by its high requirement coverage metrics performance (Table 4), the BART model generally succeeds at using input requirements in the generated tasks. Furthermore, as illustrated in Table 3, the method manages to recognize non-essential requirements and deals with them accordingly in the generated tasks (see underlined part "if using" in the generated recipe for $P2$). Additionally, the method does not violate the specification of input requirements, unlike its RecipeGPT [7] competitor (see struck out parts in Table 6).

Granularity When the level of detail for a certain input requirement is sufficient as-is for use in the tasks, the model should not further reduce it. However, the notion of granularity [38] of input requirements is important in cases where this aspect is crucial and has to be adjusted for a successful completion of tasks. Table 5 provides an example of a generated procedure showing that BART model is successfully acquiring the notion of requirement granularity, by predicting an extra task for reducing the requirement to a more fine-grained one. Notice that granularity reduction has been successfully performed despite the fact that the target product was not pre-specified (the focus was on $P2$).

Task order Performing the tasks in the correct order is also of paramount importance for successful completion of a whole procedure. Table 6 demonstrates the BART model ability to produce a proper task sequence, unlike RecipeGPT [7] which suggests making blueberry bread by adding blueberries after the bread is actually made (see underlined parts in Table 6).

Procedure conciseness An ideal procedure apart from being correct, should also be succinct. Figure 2 shows that the BART generated procedures for test set recipes contain less words in general than their ground truth counterparts (in the interest of space, we omit analogous

Table 5: An example of recipe with granularity reduction. Figure (a) displays a set of input requirements (with potato, a requirement whose granularity is of interest, underlined), (b) ground truth recipe, (c) recipe generated for $P2$ by the Transformer and (d) recipe generated for $P2$ by the BART model. Both ground truth and the BART model successfully reduce the granularity of the potato ingredient (see underlined instructions in (b) and (d)), whereas the Transformer fails to process the potato ingredient any further.

(a) Input requirements (ingredients)	(b) Ground truth procedure (verbatim from the dataset)	(c) Procedure generated by the Transformer model	(d) Procedure generated by the BART model
Requirements	**Target product** Dilled Potato-Cucumber Salad	**Target product** Creamy Potato Salad	**Target product** Creamy Potato Salad
• 1 lb new potato • 13 cup sour cream or 13 cup creme fraiche • 6 cornichons, chopped (about 1/4 cup) • 2 tablespoons fresh dill, chopped • 2 teaspoons capers, rinsed, drained and chopped • 1 seedless cucumber, thinly sliced (about 2 cups)	**Tasks** 1. Cover potatoes with cold water in pot, and bring to a boil. 2. Add a pinch of salt, reduce heat to medium, and cook 20 minutes or until tender. 3. Meanwhile, combine sour cream, cornichons, dill and capers in a bowl. 4. Drain potatoes, slice in half and return to pot. 5. Warm over medium heat 1 to 2 minutes, to dry out slightly. 6. Toss potatoes and cucumber with sour cream mixture. 7. Season with salt and pepper and serve.	**Tasks** 1. Preheat oven to 350 degrees F (175 degrees C). 2. In a large bowl, combine sour cream, sour cream, sour cream, sour cream, sour cream, and sour cream. 3. Season with salt and pepper. 4. Pour over potato mixture. 5. Cover and refrigerate for at least 1 hour. 6. Serve with sour cream.	**Tasks** 1. Peel and core potatoes; cut into 1/2-inch cubes. 2. Place in large bowl. 3. Add sour cream, cornichons, dill, capers and cucumber; toss to coat. 4. Cover and refrigerate at least 1 hour or overnight. 5. Serve at room temperature.

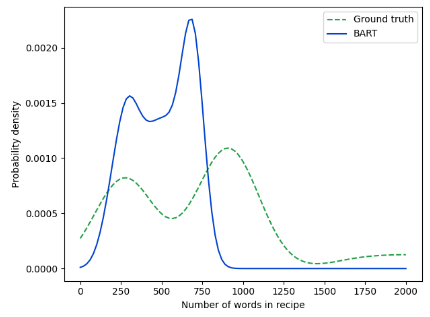

Figure 2: Comparison of recipe length distributions on test set recipes (ground truth versus BART generated).

comparison of the number of instructions distributions as they exhibit similar behavior).

7. Conclusion

In this work, we contribute to the field of procedure generation, by formalising the problem of procedure generation, positioning it from the perspective of machine translation and compiling a set of evaluation metrics which can be used to evaluate generated procedures. Most remarkably, considering the procedure generation problem as a machine translation task, allows for leveraging the novel contributions in the field of natural language generation such as BART. Using the case study from computational cooking, we showcase that these advancements enhance procedure generation in both quantitative and qualitative ways, providing correct task order, requirement coverage and granularity level, thus bringing clear advantage with respect to competing methods, including those on which RecipeGPT is based. These results can be used as a reference in future work.

To further prove the value of our approach for procedure generation, for future work we suggest additional evaluation on different datasets like wikiHow [39] (general procedural knowledge) and Pistachio [40] (chemical synthesis procedures). Other applications could include construction (combining raw materials to achieve the desired structure given upfront) or appliance

Table 6: An example of instructions generated by recipeGPT [7] and BART for $P1$. Figure (a) displays a recipe title and a set of input requirements, (b) ground truth tasks, (c) tasks generated for $P1$ by the recipeGPT and (d) tasks generated for $P1$ by BART. RecipeGPT errs twice. First (marked by striking out), it uses cinnamon, olive oil and splenda that are not planned as the input requirements. Second (wavy underline), recipeGPT fails to add the blueberries to the dough before baking, thus mistaking the task order. In contrast, BART generates a proper, ground truth-like recipe.

(a) Input: target product (recipe title) and requirements (ingredients)	(b) Ground truth tasks (verbatim from the dataset)	(c) Tasks generated by the recipeGPT model	(d) Tasks generated by the BART model
Target product Williams Sonoma Blueberry Flat Bread **Requirements** • Pizza dough (premade, your favorite kind) • 1 pint fresh blueberries • Sugar (I use splenda)	**Tasks** 1. Roll out dough to fit in regular pizza pan. 2. Sprinkle with sugar. 3. Cover with blueberries. 4. Sprinkle with sugar again. 5. cook at 375 for about 30 minutes (until crust browns and blueberries bubble). 6. Let cool; cut into pieces.	**Tasks** 1. Preheat oven to 400 degrees 2. In a small bowl, mix the sugar ~~and cinnamon together~~ 3. Roll the pizza dough into a flat circle ~~and brush with a little olive oil~~ 4. Sprinkle with ~~cinnamon/~~sugar mixture 5. Cut the dough into 12 wedges 6. Roll each wedge up from the wide end and place on a baking sheet, pointy side down 7. Bake for about 10 minutes, or until lightly browned 8. While the dough bakes, mix together 1/2 cup fresh blueberries, ~~1/4 teaspoon cinnamon, and 2 teaspoons splenda (or to taste)~~ 9. Top the warm flatbread with blueberry mixture, and serve immediately	**Tasks** 1. Preheat oven to 400 degrees. 2. Roll out pizza dough on a floured surface to a 12 inch circle. 3. Spread blueberries on dough. 4. Sprinkle sugar over blueberries. 5. Fold dough over and seal edges. 6. Bake for 15 minutes or until golden brown.

manuals (combining parts to build appliances).

References

[1] F. Stahlberg, "Neural machine translation: A review," *arXiv preprint arXiv:1912.02047*, 2019.

[2] A. Siddharthan, "A survey of research on text simplification," *ITL-International Journal of Applied Linguistics*, vol. 165, no. 2, pp. 259–298, 2014.

[3] J. Clarke and M. Lapata, "Discourse constraints for document compression," *Computational Linguistics*, vol. 36, no. 3, pp. 411–441, 2010.

[4] A. Gatt and E. Krahmer, "Survey of the state of the art in natural language generation: Core tasks, applications and evaluation," *Journal of Artificial Intelligence Research*, vol. 61, pp. 65–170, 2018.

[5] C. Kiddon, L. Zettlemoyer, and Y. Choi, "Globally coherent text generation with neural checklist models," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 329–339, 2016.

[6] S. Mori, H. Maeta, T. Sasada, K. Yoshino, A. Hashimoto, T. Funatomi, and Y. Yamakata, "Flowgraph2text: Automatic sentence skeleton compilation for procedural text generation," in *Proceedings of the 8th International Natural Language Generation Conference (INLG)*, pp. 118–122, 2014.

[7] H. H. Lee, K. Shu, P. Achananuparp, P. K. Prasetyo, Y. Liu, E.-P. Lim, and L. R. Varshney, "Recipegpt: Generative pre-training based cooking recipe generation and evaluation system," in *Companion Proceedings of the Web Conference 2020*, pp. 181–184, 2020.

[8] C. Paris, K. V. Linden, and S. Lu, "Automated knowledge acquisition for instructional text generation," in *Proceedings of the 20th annual international conference on Computer documentation*, pp. 142–151, 2002.

[9] A. Degani and E. L. Wiener, "Cockpit checklists: Concepts, design, and use," *Human factors*, vol. 35, no. 2, pp. 345–359, 1993.

[10] L. R. Varshney, F. Pinel, K. R. Varshney, A. Schörgendorfer, and Y.-M. Chee, "Cognition as a part of computational creativity," in *2013 IEEE 12th International Conference on Cognitive Informatics and Cognitive Computing*, pp. 36–43, IEEE, 2013.

[11] R. Barzilay and M. Lapata, "Modeling local coherence: An entity-based approach," *Computational Linguistics*, vol. 34, no. 1, pp. 1–34, 2008.

[12] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[14] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," 2019.

[15] A. Salvador, M. Drozdzal, X. Giro-i Nieto, and A. Romero, "Inverse cooking: Recipe generation from food images," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[16] K. R. Chandu, E. Nyberg, and A. Black, "Storyboarding of recipes: grounded contextual generation," 2019.

[17] K. J. Hammond, "Chef: A model of case-based planning.," in *AAAI*, pp. 267–271, 1986.

[18] R. Barzilay and M. Lapata, "Collective content selection for concept-to-text generation," in *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pp. 331–338, 2005.

[19] B. P. Majumder, S. Li, J. Ni, and J. McAuley, "Generating personalized recipes from historical user preferences," *arXiv preprint arXiv:1909.00105*, 2019.

[20] P. Pareti, E. Klein, and A. Barker, "Linking data, services and human know-how," in *European Semantic Web Conference*, pp. 505–520, Springer, 2016.

[21] P. Chocron and P. Pareti, "Vocabulary alignment for collaborative agents: a study with real-world multilingual how-to instructions.," in *IJCAI*, pp. 159–165, 2018.

[22] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *CoRR*, vol. abs/1409.3215, 2014.

[23] D. Jurafski, "Cs 124: From languages to information." `http://web.stanford.edu/class/cs124/lec/languagemodeling.pdf`, 2019. Accessed May 14, 2020.

[24] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting on association for computational linguistics*, pp. 311–318, Association for Computational Linguistics, 2002.

[25] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*, (Barcelona, Spain), pp. 74–81, Association for Computational Linguistics, July 2004.

[26] T. Zhang*, V. Kishore*, F. Wu*, K. Q. Weinberger, and Y. Artzi, "Bertscore: Evaluating text generation with bert," in *International Conference on Learning Representations*, 2020.

[27] H. Li, "Learning to rank for information retrieval and natural language processing," *Synthesis Lectures on Human Language Technologies*, vol. 4, no. 1, pp. 1–113, 2011.

[28] L. A. M. Bostan, "Ingredient-driven recipe generation using neural and distributional models." `https://lct-master.org/getfile.php?id=2194&n=1&dt=TH&ft=pdf&type=TH`, 2017. Accessed May 14, 2020.

[29] J. Marin, A. Biswas, F. Ofli, N. Hynes, A. Salvador, Y. Aytar, I. Weber, and A. Torralba, "Recipe1m+: A dataset for learning cross-modal embeddings for cooking recipes and food images," *IEEE Trans. Pattern Anal. Mach. Intell.*, 2019.

[30] P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, *et al.*, "Moses: Open source toolkit for statistical machine translation," in *Proceedings of the 45th annual meeting of the association for computational linguistics companion volume proceedings of the demo and poster sessions*, pp. 177–180, 2007.

[31] M. Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, and M. Auli, "fairseq: A fast, extensible toolkit for sequence modeling," in *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

[32] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," *CoRR*, vol. abs/1508.07909, 2015.

[33] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[34] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[35] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[36] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning," *CoRR*, vol. abs/1705.03122, 2017.

[37] D. C. J. G. Changhan Wang, Anirudh Jain, "Vizseq: A visual analysis toolkit for text generation tasks," in *In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2019.

[38] P. Schumacher, *Workflow extraction from textual process descriptions*. Verlag Dr. Hut München, 2016.

[39] P. Pareti, B. Testu, R. Ichise, E. Klein, and A. Barker, "Integrating know-how into the linked data cloud," *CoRR*, vol. abs/1604.04506, 2016.

[40] J. Mayfield, D. Lowe, and R. Sayle, "Pistachio. 2.0." `https://www.nextmovesoftware.com/pistachio.html`, 2018.