10-2022

Accurate generation of trigger-action programs with domain-adapted sequence-to-sequence learning

IMAM NUR BANI YUSUF
Singapore Management University, imamy.2020@phdcs.smu.edu.sg

Lingxiao JIANG
Singapore Management University, lxjiang@smu.edu.sg

David LO
Singapore Management University, davidlo@smu.edu.sg

Citation

IMAM NUR BANI YUSUF; JIANG, Lingxiao; and LO, David. Accurate generation of trigger-action programs with domain-adapted sequence-to-sequence learning. (2022). *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension, ICPC 2022, Virtual Event, May 16-17, 2022*. 99-110.
Available at: https://ink.library.smu.edu.sg/sis_research/7723

Accurate Generation of Trigger-Action Programs with Domain-Adapted Sequence-to-Sequence Learning

Imam Nur Bani Yusuf, Lingxiao Jiang, David Lo
School of Computing and Information Systems
Singapore Management University, Singapore
imamy.2020@phdcs.smu.edu.sg,lxjiang@smu.edu.sg,davidlo@smu.edu.sg

ABSTRACT

Trigger-action programming allows end users to write event-driven rules to automate smart devices and internet services. Users can create a trigger-action program (TAP) by specifying triggers and actions from a set of predefined functions along with suitable data fields for the functions. Many trigger-action programming platforms have emerged as the popularity grows, e.g., IFTTT, Microsoft Power Automate, and Samsung SmartThings. Despite their simplicity, composing trigger-action programs (TAPs) can still be challenging for end users due to the domain knowledge needed and enormous search space of many combinations of triggers and actions. We propose RecipeGen, a new deep learning-based approach that leverages Transformer sequence-to-sequence (seq2seq) architecture to generate TAPs on the fine-grained field-level granularity from natural language descriptions. Our approach adapts autoencoding pre-trained models to warm-start the encoder in the seq2seq model to boost the generation performance. We have evaluated RecipeGen on real-world datasets from the IFTTT platform against the prior state-of-the-art approach on the TAP generation task. Our empirical evaluation shows that the overall improvement against the prior best results ranges from 9.5%-26.5%. Our results also show that adopting a pre-trained autoencoding model boosts the MRR@3 further by 2.8%-10.8%. Further, in the field-level generation setting, RecipeGen achieves 0.591 and 0.575 in terms of MRR@3 and BLEU scores respectively.

CCS CONCEPTS

• **Software and its engineering → Search-based software engineering**; • **Information systems → Retrieval models and ranking**; • **Computing methodologies → Machine translation**.

KEYWORDS

Trigger-Action Programming, IFTTT, Program Generation, Deep Learning, Encoder-Decoder

ACM Reference Format:
Imam Nur Bani Yusuf, Lingxiao Jiang, David Lo. 2022. Accurate Generation of Trigger-Action Programs with Domain-Adapted Sequence-to-Sequence Learning . In *30th International Conference on Program Comprehension (ICPC '22), May 16–17, 2022, Virtual Event, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3524610.3527922

1 INTRODUCTION

End-user programming is to make computer programming more accessible to all people, especially those who have a background outside of Computer Science [1]. One approach for this is to create a higher-level abstraction to hide the low-level detail of a programming language. In this work, we are interested in a simple yet useful type of programming called trigger-action programming.

Trigger-action programming allows end-users to write event-driven rules to automate smart devices and internet services [2]. Users can create a Trigger-Action Program (or simply TAP) by specifying some triggers and actions from a set of predefined trigger and action functions. For example, a TAP can be expressed as an if-then conditional rule as follows, "IF trigger condition is satisfied, THEN execute the action." Many trigger-action programming platforms have emerged as the popularity of IoT (Internet of Things) grows, such as IFTTT [3], Olisto [4], Integromat [5], Zapier [6], Microsoft Power Automate [7], Home Assistant [8], and OpenHAB [9].

Writing TAPs can still be challenging for end-users due to the enormous search space and the domain knowledge needed for all the triggers and actions. To exemplify, the number of available services (e.g., Gmail, DropBox, Philips Hue Bulb, WeMo Smart Plug) in IFTTT by 2017 was 408 [10], resulting in 468,930 possible combinations of trigger and action functions. By June 2021, the number increases to 1,386 [3]. The study conducted by Corno et al. [11] shows that users may not be aware of the usable trigger and action functions due to the many services available.

We focus on TAPs from the IFTTT platform in this work due to the following reasons. First, it is popular and has a large user base [10, 12]. Second, it supports more than 1300 channels as of June 2021. Last, a few prior studies on IFTTT [10, 13] have made some IFTTT datasets publicly available and it has been widely used on other TAP generation studies [14, 15]. Our study can be applied to other platforms even though we focus on a specific platform because the other aforementioned platforms work similarly as IFTTT; users compose TAPs by specifying triggers followed by actions.

The current state-of-the-art on automatic TAP generation is Latent Attention Model (LAM) [14]. LAM takes a natural language description of the intended TAP as the input, and identifies the required trigger and action functions to form the target TAP. LAM frames the problem as a multi-class classification task, where a number of disjoint classifiers are trained separately to classify the trigger and action to which the input description may belong. LAM performs poorly when an input description does not describe the intended functionality explicitly, or when the description is vague or incomplete even though it outperforms prior approaches [13, 15]. We refer to such a description as an *unclear description*. An example of unclear descriptions is shown in Figure 1. The first description mentions "photo tagged with me" in the context of "fb" and "dropbox"; it does not explicitly mention "file" or any function of the Facebook and Dropbox. Given the description, a TAP generation model thus needs to learn implicit relations between "photo" and "file", between "me" and "you", between "Facebook"/"Dropbox" and "url", in order to identify the right functions to use in a TAP. Learning to classify the trigger and action functions *disjointedly* hinders LAM from leveraging many implicit relations between the trigger and action functions, limiting its TAP generation performance, especially when the input descriptions are unclear.

Furthermore, when a user uses a function to compose a TAP (e.g., in Figure 1), the function may require the user to select some field names and fill in those field names with some values. For instance, Dropbox.Add_File_From_Url requires the user to specify the URL source of the image in "File URL", the name of the file in "File Name", and the target folder in "Folder Path". These field names are predefined and can be either a required one or an optional one. Specifying and filling in the field names can be challenging for end-users due to the domain-knowledge needed. Another limitation of prior studies [13, 15, 14] is that they only generate the channels and functions for a given description (which we call *function-level* generation). Our approach also addresses this problem, aiming to generate TAPs with the needed field names too (which we call *field-level* generation). Generating the field names automatically can lessen the burden of end-users as they do not need to worry about the background knowledge needed to determine the relevant field names based on the given description and the selected function.

This paper presents RecipeGen, a deep learning-based approach that allows users to automatically generate TAPs using natural language descriptions. Different from prior studies which formulate TAP generation as a classification task [13, 15, 14], RecipeGen frames the problem as a sequence learning and generation task. RecipeGen leverages Transformer sequence-to-sequence (seq2seq) architecture [16] to translate an input description into a sequence of triggers and actions. We demonstrate that framing the problem as sequence learning is better than classification because sequence learning allows the model to leverage implicit relations between the channels, functions, and fields of triggers and actions.

To boost the generation performance, RecipeGen also adapts an autoencoding pre-trained model to initialize and warm-start the parameters of the encoder in the seq2seq model. The intuition of instantiating the encoder using a pre-trained autoencoding model is inspired by the fact that the model has been pre-trained using data-rich corpora such that it has a sense of understanding the meaning of input descriptions about TAPs even though those data are not for TAPs. Intuitively, the learning burden of a seq2seq model can be reduced if the encoder can have such knowledge of the words and phrases that may appear in TAPs and their descriptions at the beginning of its training.

We have evaluated RecipeGen against the prior state-of-the-art LAM [14], on 4 real-world datasets from the IFTTT platforms. Our results demonstrate that RecipeGen, leveraging sequence learning, consistently outperforms LAM in all datasets in the function-level generation. Specifically, RecipeGen achieves 0.947 on gold-standard

Description from a User
Saves any photo tagged with me in it on fb to Dropbox
TAP
Trigger : Facebook.You_are_Tagged_in_Photo
Field(s) :
- None

Action : Dropbox.Add_File_From_Url
Field(s) :
- File URL
- File Name
- Folder Path

Figure 1: An example of TAP and its description.

test set and 0.647 on noisy test set in terms of MRR@3. Overall, the improvement results against LAM ranges from 9.5%-26.5%. Our results also shows that adopting a pre-trained autoencoding model boosts the MRR@3 further by 2.8%-10.8%. Further, in the field-level generation setting that LAM does not address, RecipeGen achieves 0.591 and 0.575 in terms of MRR@3 and BLEU score.

The main contributions of our study are the following.

- We formulate TAP generation as a sequence learning task and propose RecipeGen, a deep learning-based approach that adapts Transformer seq2seq architecture to address the task. RecipeGen can generate TAPs at the fine-grained field-level which the prior approaches [13, 15, 14] do not address. We make the replication package of our study, including its source code and datasets for evaluation, publicly available[1].
- We adapt existing off-the-shelf pre-trained autoencoding models to boost the performance of RecipeGen. We also scrutinize various factors that can affect the efficiency of such adaptation.
- We have evaluated our approach on real-world datasets from the IFTTT platform. Our results demonstrate that RecipeGen achieves better performance than the prior state-of-the-art in both function-level and field-level generation settings.

The rest of the paper is structured as follows. Section 2 covers the related backgrounds on TAP and Transformer. Section 3 explains our problem formulation and our proposed approach. Section 4 describes the experimental settings. Section 5 presents the empirical results, whereas Section 6 presents the error analysis and the various threats to the validity of our study. Section 7 presents related works in more details. Section 8 concludes our paper.

2 PRELIMINARY

2.1 Trigger Action Program (TAP)

A basic TAP is composed of a trigger and an action expressed in the form of an if-then conditional rule; "IF trigger condition is satisfied, THEN execute the action." TAP allows users to build integration between internet services and/or smart devices [13].

A sample snippet of TAP from IFTTT paired with its natural language description is shown in Figure 1. The snippet contains one trigger and one action. Each trigger and action has a channel and a function component. A channel is an entity that provides various services. A function is an API that represents a particular service or channel's functionality. A function may require the user to specify some fields from a set of predefined fields to control how

[1]https://github.com/imamnurby/RecipeGen-IFTTT-RP

Table 1: Grammar for a TAP

$$
\begin{aligned}
recipe &::= \langle triggerList \rangle \ \langle actionList \rangle \\
triggerList &::= \langle trigger \rangle \mid \langle trigger \rangle \ \langle triggerList \rangle \\
actionList &::= \langle action \rangle \mid \langle action \rangle \ \langle actionList \rangle \\
trigger &::= \langle trigChannel \rangle \ \langle trigFunction \rangle \\
action &::= \langle actChannel \rangle \ \langle actFunction \rangle \\
trigFunction &::= \langle functionName \rangle \ \langle fieldList \rangle \\
actFunction &::= \langle functionName \rangle \ \langle fieldList \rangle \\
fieldList &::= \langle null \rangle \mid \langle field \rangle \ \langle fieldList \rangle
\end{aligned}
$$

exactly the function work. A field is composed of a field name and a field value. A field can be either a required one or an optional one. We formalize the representation of a TAP using the grammar shown in Table 1.

The TAP in Figure 1 leverages "Facebook" and "Dropbox" as the trigger channel and action channel respectively. The trigger function is "Facebook.You_are_Tagged_in_Photo" and it does not have any fields. The action function is "Dropbox.Add_File_From_Url" and it has three fields, i.e., "File URL" that specifies the source of the file that will be uploaded, "File name" that specifies the name of the file, and "Folder path" that specifies the upload destination. In IFTTT, users can write TAP in Figure 1 by specifying each component *manually* through a web-based graphical user interface, and users may spend sometime before finding the intended channels or functions due to the large search space [11, 17].

2.2 Transformer-Based Language Model

Transformer seq2seq architecture [16] is illustrated in Figure 2. A Transformer seq2seq model is composed of encoder and decoder blocks. Given a description, the tokenizer converts the description into a sequence of token-ids $x_1, x_2, ..., x_n$. The encoder embedding then converts the sequence of token-ids into the input vectors $x'_1, x'_2, ..., x'_n$. The encoder takes the input vectors and feeds them to the Self-Attention layer. The output of the encoder is context vectors $c_1, c_2, ..., c_n$. The decoder follows similar steps but has two differences, i.e., 1) the inputs are the context vectors $c_1, c_2, ..., c_n$ and the prior predictions $y_1, y_2, ..., y_{t-1}$, and 2) it uses Causal-Attention layer instead of Self-Attention layer. The output of the decoder is then converted into the probability distribution over the target vocabularies using a linear layer called LM Head.

The difference between Self-Attention layer and Causal-Attention layer is on how the model attends to the tokens. Self-Attention layer allows a token at position i, i.e., x_i, to attend to both its preceding tokens $x_1, x_2, ..., x_{i-1}$ and its succeeding tokens $x_{i+1}, x_{i+2}, ..., x_n$. In contrast, Causal-Attention layer only allows x_i to attend to its preceding tokens $x_1, x_2, ..., x_{i-1}$.

With the progress of transfer learning [18, 19, 20], the research direction moves towards building a pre-trained model using Transformer architectures. A pre-trained model refers to a model that has been trained on large corpora with certain loss functions. The idea is to transfer knowledge learned from data-rich corpora in the pre-training phase to downstream tasks, which often have much less data available for training.

Many existing pre-trained models adapt Transformer architectures [16]. These models can be classified into three categories [21], i.e., autoencoding, autoregressive, and seq2seq. Each category is

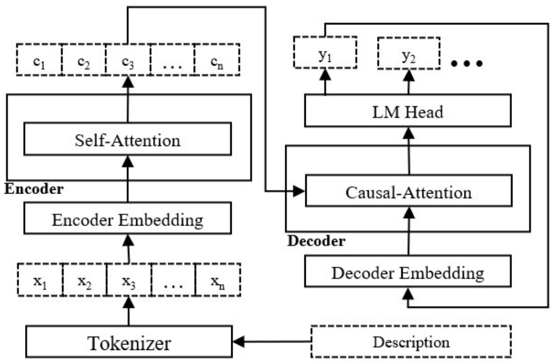

Figure 2: Transformer seq2seq architecture (simplified). The dashed line indicates inputs and outputs, while the bold line indicates the model's components.

to address different downstream tasks. Autoencoding models are composed of *only* the encoder of the seq2seq architecture shown in Figure 2; these models only leverage the Self-Attention layer, and are suitable for downstream tasks that require an understanding of the whole context, e.g., defect prediction and clone detection [22]. Autoregressive models are composed of *only* the decoder of the seq2seq architecture; these models only leverage the Causal-Attention layer, and are suitable for downstream generative tasks such as code completion [23]. Seq2seq models leverage both the encoder and decoder blocks [21], and are suitable for converting a sequence to another sequence, e.g., code generation, code summarization, and code repair [24].

3 APPROACH

This section covers two parts: 1) problem formulation and 2) the workflow of RecipeGen. The first part covers the formal definition of sequence learning in TAP and discusses why sequence learning is well-suited for TAP generation. The second part explains the workflow of RecipeGen that consists of model initialization, training stage, and inference stage.

3.1 Problem Formulation

We formulate the problem based on the fact that users follow a particular event *sequence* when composing a TAP in a real-world implementation. Users select a trigger *followed by* an action. Users should specify the channel *followed by* the function and its fields for each trigger and action. In other words, functions are selected based on the specified channels, and fields are selected based on the specified functions and channels. We formalize TAP generation process as follows,

$$y = \underset{y}{\operatorname{argmax}} \ P(y_t \mid y_{<t}, I) \tag{1}$$

where y_t denotes the probability distribution over the target components (i.e., channels, functions, and fields) at the time step t, $y_{<t}$ is the sequence of the prior predictions, and I is the input description. Equation 1 illustrates that the prediction for the current time step t is based on the sequence of the predictions at the previous time steps ($y_{<t}$) and the input description I. Such a problem formulation conforms to the steps of composing TAP in a real-world

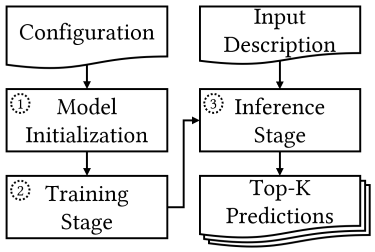

Figure 3: RecipeGen's workflow.

implementation. We believe that sequence learning is a better formulation than classification-based approaches proposed in the prior studies [13, 14, 15] because sequence learning allows the model to capture the implicit conditional relationship between the target components, and it is easily applicable to both function-level and field-level generation settings, by adjusting the number of time steps used. In contrast, the prior approaches [13, 14, 15] do not consider $y_{<t}$; they generate each trigger and action disjointedly for a given input description, and do not generate field names.

3.2 RecipeGen

The workflow of RecipeGen is shown in Figure 3. RecipeGen has three working stages: model initialization, training, and inference. In the model initialization stage, RecipeGen instantiates a seq2seq model (see Figure 2) and initializes the encoder with a pre-trained autoencoding model. In the training stage, RecipeGen trains the model to translate descriptions in natural language into target TAPs. In the inference stage, RecipeGen is ready to generate a TAP: it returns a ranked list containing the most likely top-K TAPs for a given description describing the functionality of the intended TAP.

3.2.1 Model Initialization. The input of this stage is a configuration file and the output is an initialized but untrained seq2seq model. First, RecipeGen instantiates the architecture of the encoder embedding and the encoder block according to the settings in the configuration file, and loads the model's vocabularies and the pre-trained embedding weights, and use these to initialize the encoder embedding. RecipeGen also loads the tokenizer and configures it using the model's vocabularies. Second, RecipeGen loads the pre-trained weights of the encoder, then use these weights to instantiate the encoder block. Third, RecipeGen instantiates the architecture of the decoder block using the same decoder as the decoder of Transformer seq2seq architecture [16], then initializes the weights randomly. For the decoder embedding, RecipeGen leverages the same weights as the encoder embedding to allow the model to leverage the similarity between the descriptions and the TAP components. We observe that the TAP components often resemble a normal natural language and share the same subwords as the descriptions. For instance, the description in Figure 1 contains two subwords "photo" and "tagged" of the target trigger function and another subword "Dropbox" of the target action channel.

3.2.2 Training Stage. To exploit the similarity between the descriptions and the TAPs at the subword-level, we train RecipeGen to generate TAPs at the subword-level.

The tokenizer converts each word in a description to the corresponding token ids. For TAP-specific words (e.g., channel names)

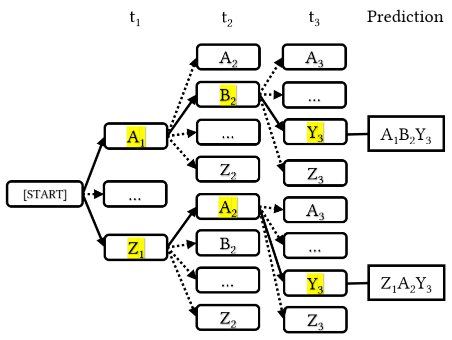

Figure 4: An example of beam search with beam width=2. The bold line indicates the actual expansion.

that do not exist in the vocabulary, the tokenizer breaks down such words into a sequence of valid *subwords* that exist in the vocabulary, where the smallest unit can be a letter. For example, "Saves any photo to dropbox" can be tokenized into "S", "aves", "any", "photo", "to", "drop", "box". Then, each token is converted to the corresponding token-ids, and is subsequently converted into a vector by passing the token-ids through the encoder embedding.

The output of the model is represented as an ordered sequence of 6 components, i.e., TC, TF, FN_{TF}, AC, AF, FN_{AF}, where TC is a trigger channel, TF is a trigger function, FN_{TF} is a sequence of trigger field names, AC is an action channel, AF is an action function, and FN_{AF} is a sequence of action field names. FN_{TF} and FN_{AF} can be empty if the function does not have any field names. To ensure RecipeGen knows to which component a subword belongs, we insert a special token [SEP] to indicate the beginning of a trigger channel, a trigger function, an action channel, and an action function. We also insert [(] and [)] to indicate the beginning and end of the field names respectively, inside the trigger and action functions. Lastly, we use [END] to indicate the end of a TAP. The following sequence illustrates the output of the decoder block in the training stage: [SEP] Facebook [SEP] Facebook.You_are_Tagged_in_a_Photo [(] [)] [SEP] Dropbox [SEP] Dropbox.Add_File_from_Url [(] File URL, File Name, File Path [)] [END]. These special tokens later are omitted in the final output.

As explained in the prior stage, RecipeGen outputs the prediction at the subword-level. The TAP in Figure 1 is generated as follows[2].

TC	[SEP]#Facebook#
TF	[SEP]#Facebook#.#You#_#Are#_#T#agged#_#in#_#Photo#
FN_{TF}	[(]#[)]#
AC	[SEP]#Dropbox#
AF	[SEP]#Dropbox#.#Add#_#File#_#From#_#URL#
FN_{AF}	([(]#File# #URL#,#File# #Name#,# #File# #Path#[)]#[END]#

3.2.3 Inference Stage. RecipeGen leverages beam search [25] to generate the target TAP. A sample beam search with beam width 2 is shown in Figure 4. The first time step selects two candidates: A_1 and Z_1 as these two have the highest probability; each candidate in our context can be a subword for a channel, a function, or a field name, depending on the current time step. At the subsequent time step (t_2), A_1 can be expanded to $A_1\#A_2$, $A_1\#B_2$, ... $A_1\#Z_2$. The same applies to the other candidate Z_1. The probability of each

[2]RecipeGen does not generate "#" in the actual generation; it is only used to indicate the generated token in each time step.

Table 2: Dataset Statistics

Statistics	Train				Validation		Test		
	Quirk15	Mi17	Merged	Train+Field	Val15	Val+Field	Gold15	Noisy15	Test+Field
#Samples	45,003	107,008	120,794	95,087	1,451	5,283	305	769	5,283
#Trigger channels	112	311	327	307	71	184	51	66	179
#Trigger functions	492	1.040	1.107	1014	211	468	112	170	468
#Trigger fields	-	-	-	338	-	178	-	-	182
#Avg. fields/trig. function	-	-	-	1.10	-	1.10	-	-	1.10
#Action channels	87	282	288	275	57	170	38	48	164
#Action functions	201	665	687	650	106	296	60	90	279
#Action fields	-	-	-	415	-	244	-	-	261
#Avg. fields/act. function	-	-	-	2.31	-	2.33	-	-	2.31

expansion candidate is computed by multiplying the probability of the prior candidates with the current candidates; e.g., the probability of $A_1\#A_2$ is obtained by multiplying the probability of A_1 and A_2 conditioned on the A_1 generated at t_1. The expansion is performed by taking the top-K candidates with the largest probability, where K equals to the beam width size. The expansion for a candidate stops if the model has reached the maximum allowed expansion length or has generated the special token that indicates the end of a sequence.

4 EVALUATION

4.1 Dataset

Our study leverages datasets published by Quirk et. al. [13] and Mi et. al. [10]. Both datasets are curated from the IFTTT platform. We perform evaluation using four train sets, two validation sets, and three test sets derived from those datasets. Table 2 presents the overview of the train, validation, and test sets.

4.1.1 Train Set. We leverage four train sets, i.e., Quirk15, Mi17, Merged, and Train+Field.

- Quirk15 is comprised of 45K TAPs paired with its descriptions curated by Quirk et. al. [13]. The number of channels and functions is the least among all the train sets.
- Mi17 contains 107K TAPs paired with its descriptions curated by Mi et. al. [10]. This dataset is more recent than Quirk15 and contains more diverse channels and functions than Quirk15.
- Merged dataset consists of 120K TAPs and its descriptions, obtained by merging Quirk15 and Mi17 datasets. We merge both datasets to get the maximum number of samples, channels, and functions. The number of samples is less than the sum of Quirk15 and Mi17 because there is an intersection between both datasets.
- Train+Field is similar to Mi17, but it contains additional information about field names. Mi et. al. [10] have prepared their dataset such that the field names can be obtained easily. We split Mi17 into Train+Field, Val+Field, and Test+Field. We leverage a stratified sampling strategy in the split process, such that descriptions with different token lengths are equally distributed among the splits. The number of fields is often smaller than the number of functions because some functions share similar field names.

Table 3 shows some sample TAPs and their descriptions from the datasets. The first and second examples illustrate unclear descriptions. Example (1) only mentions the channels' names without describing the specific functionalities for the trigger and the action. Inferring the functions is difficult in such a case because a channel can have several functionalities and these functions may also have high similarity, e.g., Twitter.Post_a_Tweet and Twitter.Post_a_Tweet_with_image. Example (2) describes the functionality without mentioning the channels. Such a case can also be difficult because different channels may have the same functionality, e.g., WeMo_Switch.Turn_on and WeMo_Smart_Plug.Turn_on.

The examples (3) and (4) illustrate clear descriptions. Both descriptions mention the channel names and their functionality clearly. However, we can observe that inferring the field names and values are challenging in such descriptions because the possible values can be very sparse and unique for each use case. For example, inferring the field value for "Select a category" in the example (4) is difficult because the description does not contain any relevant information. On the other hand, generating the field names is still possible because field names are predefined and multiple TAPs may use the same trigger or action function with the same fields resulting in some reusable patterns. We only generate field names and leave the generation of field values as future work.

4.1.2 Validation Set. We use two validation sets, i.e., Val15 and Val+Field.

- Val15 set contains 1.4K TAPs paired with its descriptions. Val15 is originally created by Quirk et. al. [13], and is used to measure how well the model performs under the function-level generation setting in the training stage.
- Val+Field set contains 5K descriptions and TAPs along with the field names information. Val+Field is used to measure how well the model performs under the field-level generation setting in the training stage.

4.1.3 Test Set. We use three test sets in our evaluation, i.e., Gold15, Noisy15, and Test+Field. Gold15 and Noisy15 test sets are originally created by Quirk et. al. [13] through a human-annotation process.

- Gold15 set contains 305 TAPs with its gold-standard descriptions. The annotation process is done by asking Amazon Mechanical Turk workers to label each description with the corresponding TAP implementation. The workers label a description that does not contain enough information to infer the corresponding TAP with *unintelligible*. A pair of TAP and its description is included in the Gold15 set if the pair has at least three workers who agree to the label. Gold15 set is used to evaluate the model on the ideal condition where a description contains enough information to infer both trigger and action correctly.

Table 3: Examples of Descriptions and the Corresponding TAPs

	Description	Trigger	Action
(1)	Morning Android weather	Channel: Weather. Function: Today's_weather_report Fields: Time of day	Channel: Notifications Function: Send_a_notification_from_the_IFTTT_app Field: Notification
(2)	New photos to drive	Channel: iOS_Photos Function: Any_new_photo Fields: -	Channel: Google_Drive Function: Upload_file_from_URL Fields: File URL, File name, Drive folder path
(3)	Ask Alexa to "trigger red lights" to change phillips hue bulbs to red	Channel: Amazon_Alexa. Function: Say_a_specific_phrase Field: What phrase?	Channel: Philips_Hue Function: Change_color Fields: Which lights?, Color value or name
(4)	If task is completed in todoist, save it as a note in evernote notebook	Channel: NPR Function: New_story_published Field: Select a category	Channel: Evernote Function: Create_a_note() Field: Title, Body, Notebook, Tags

- Noisy15 set contains 769 TAPs and its descriptions that undergo the same labeling process as the Gold15 set. A pair of TAP and its description is included in the Noisy15 test set if it is not labeled as *unintelligible* and has less than three workers who agree to the label. Noisy15 set is used to mimic a condition where a user does not know exactly the target channel or function. Both Gold15 and Noisy15 sets are on the function-level granularity.
- Test+Field contains 5K descriptions and TAPs along with the field names information. Test+Field is used to evaluate the actual performance of the model in the field-level granularity.

4.2 Experimental Setting

4.2.1 Models. RecipeGen leverages two pre-trained autoencoding models to initialize the encoder weights, i.e., CodeBERT and RoBERTa, to compare the performance of a model that is specifically pre-trained on code corpora and a model that is pre-trained using general English corpora for the TAP generation task. We choose CodeBERT and RoBERTa as both models have been widely used in various studies [26, 27, 28].

We use LAM as the baseline. For RecipeGen, we instantiate three different model variants, i.e., Rand2Rand, Rob2Rand, and Code2Rand. Each model is described below.

- **Rand2Rand.** Rand2Rand is a Transformer seq2seq model where the weights on both the encoder and decoder are randomly-initialized.
- **Rob2Rand.** Rob2Rand is a Transformer seq2seq model that leverages RoBERTa[3] to initialize the weights of the encoder, while the weights of the decoder is initialized randomly.
- **Code2Rand.** Code2Rand is similar with Rob2Rand, but it leverages CodeBERT[4] instead of RoBERTa.
- **LAM.** LAM[5] is the prior state-of-the-art model on TAP generation task. We have faithfully migrated and reimplemented LAM which was originally implemented in TensorFlow v0.7 to TensorFlow v2.5 due to obsolete dependencies.

The hyperparameters used in our evaluation are as follows.

- **RecipeGen.** The batch size is set to 8 for training, validation, and testing. The default learning rate is 5x10^{-5}, except for Rand2Rand that is 5x10^{-6}. We use a smaller learning rate, for Rand2Rand because the models failed to learn when using the default learning rate. We use AdamW[6] as the optimizer with 1000 warming steps. For the inference, we set the beam width to 10.
- **LAM.** We use the same hyperparameters and training setting as described in the original paper [14]. The optimizer is Adam[7] with a learning rate 1x10^{-3} and the embedding dimension is 50.

We apply early stopping to prevent overfitting when training both RecipeGen and LAM.

4.2.2 Evaluation Setting. Our evaluation setting is divided into function-level and field-level. Each setting is described as follows.

- **Function-level.** We train each model on Quirk15, Mi17, Merged training sets separately and we validate the model using Val15 set. The evaluation is performed on both Gold15 and Noisy15 sets.
- **Field-level.** We train all models except LAM using Train+Field set. We use Val+Field and Test+Field to validate and test each model respectively. We do not train LAM in this setting because LAM is not designed to generate fields.

For each train, validation, and test set, the descriptions are lowercased and non-ASCII characters are removed. Non-English descriptions are also omitted. We use langdetect[8] to detect the language of each description. We run all experiments on a computer running Ubuntu 18.04 with Intel(R) Core(TM) i7-10700K CPU @ 3.80GHz processor, 64GB RAM, and NVIDIA GeForce RTX 3070 8GB.

4.3 Metrics

We evaluate the performance of all models across different datasets using Mean Reciprocal Rank (MRR) and BLEU score.

Mean Reciprocal Rank (MRR) is the average of reciprocal ranks of results from a set of queries Q [29]. MRR has been widely used to evaluate systems that return a ranked list as the output [30, 31, 32, 29]. MRR@k is computed using Equation 2:

$$MRR@k = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i} \tag{2}$$

[3]https://huggingface.co/roberta-base
[4]https://huggingface.co/microsoft/codebert-base
[5]https://github.com/Jungyhuk/Latent-Attention
[6]https://huggingface.co/docs/transformers/main_classes/optimizer_schedules#transformers.AdamW
[7]https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam
[8]https://pypi.org/project/langdetect/

Table 4: Improvement of MRR@3 on Gold15 and Noisy15. For each test set (Gold15 and Noisy15), we compare all RecipeGen variants against LAM and Rand2Rand (R2R). The bold numbers denotes the highest value.

Model	Train Set	MRR@3 Improvement (%)			
		Gold15		Noisy15	
		LAM	R2R	LAM	R2R
Rand2Rand	Merged	9.5	-	20.8	-
Rob2Rand		13.3	3.6	24.4	3.6
Code2Rand		12.8	3.2	24.2	3.4
Rand2Rand	Mi17	11.2	-	22.7	-
Rob2Rand		14.6	2.8	**26.5**	3.8
CodeBERT		15.7	4.0	26.4	3.7
Rand2Rand	Quirk15	9.5	-	16.1	-
Rob2Rand		**20.6**	**10.8**	24.2	**8.1**
Code2Rand		18.6	8.9	23.6	7.5

where rank$_i$ indicates the first rank of the first hit or relevant result. The value of $\frac{1}{rank_i}$ is 0 if the ground truth is not in the top-K returned TAPs. MRR@K ranges from 0 to 1. From a user's point of view, the higher the MRR is, the better, as the reciprocal of MRR indicates the average number of predictions that a user needs to investigate to find the correct one. We only report the result at k=3 due to the page limitation[9].

BLEU score is the hits of n-grams of translation results with its ground truths [33]. BLEU score has been widely used to evaluate the performance of various translation systems on software engineering [34, 35, 36]. BLEU score is computed using Equation 3:

$$BLEU = BP \exp \sum_{n=1}^{N} w_n \log p_n \tag{3}$$

where p_n is the n-grams precision, N is the maximum number of grams, and BP is the penalty score for short translation results. BP equals to 1 if the length of translation results c is longer than the ground truth r; otherwise, it is equals to $\exp^{(1-\frac{r}{c})}$. BLEU score ranges from 0 to 1 and we report the result at N=4 (i.e., BLEU-4). Higher BLEU score is better; it means that the translation result is more similar to the ground truth.

5 EMPIRICAL RESULTS

5.1 How does the Transformer seq2seq-based models perform on TAP generation task?

5.1.1 Function-level Evaluation. Formulating the problem as a sequence learning task yields better performance than as a classification task. Figure 5 shows that all variants of RecipeGen (i.e., Rand2Rand, Rob2Rand, and Code2Rand) trained across different datasets consistently outperform LAM on both Gold15 and Noisy15 sets in terms of MRR@3 and BLEU score. The results in Table 4 also illustrates that all variants of RecipeGen have significant improvements against LAM in terms of MRR@3. The improvements of all variants of RecipeGen over LAM range from 9.5%-20.6% on Gold15 set and 16.1%-26.5% on Noisy15 set. These results indicate that RecipeGen can deal with unclear descriptions better than LAM.

[9]We attach more comprehensive results using different values of k in our replication package at https://github.com/imamnurby/RecipeGen-IFTTT-RP

Table 5: TAP generation results on field-level. Each model is trained on the Train+Field set and evaluated on the Test+Field set. R2R indicates Rand2Rand. The bold numbers denotes the highest value.

Models	MRR@3	BLEU-4	Improvement over R2R	
			MRR@3 (%)	BLEU-4 (%)
Rand2Rand	0.555	0.536	-	-
Rob2Rand	0.588	0.571	3.3	3.5
Code2Rand	**0.591**	**0.575**	**3.6**	**3.9**

Takeaway 1: All variants of RecipeGen trained across different dataset consistently outperform LAM on both Gold15 and Noisy15 sets in terms of MRR@3 and BLEU score. The improvement of all variants of RecipeGen over LAM ranges from 9.5%-20.6% on Gold15 set and 16.1%-26.5% on Noisy15 set.

Both Rob2Rand and Code2Rand perform better than Rand2Rand on both Gold15 and Noisy15 sets, as shown in Figure 5, although the performance improvement is sometimes small. Table 4 illustrates that both Rob2Rand and Code2Rand that are trained on Mi17 and Merged sets have lower improvement than models that are trained on Quirk15 set on both Gold15 and Noisy15 sets. For instance, Rand2Rand trained on Quirk15 achieves 0.812 in terms of MRR@3 (see Figure 5, top-left). When Rand2Rand trained on Merged set, the MRR@3 increases significantly to 0.909. The evaluation on Noisy15 set (see Figure 5, bottom-left) also shows the same trend.

The implications of such results are the following. Firstly, the diversity of the channels and functions does not hurt the performance because both Mi17 and Merged sets have a higher number of channels and function than Quirk15 and our results show that the performance of the models trained either on Mi17 or Merged sets are better than the models trained on Quirk15. Secondly, the improvement of using a pre-trained autoencoding model is higher in the limited data setting as Quirk15 set has much fewer samples than both Mi17 and Merged set. Table 4 shows that the overall MRR@3 improvement is higher on both Gold15 and Noisy15 sets when Quirk15 is used as the train set.

Takeaway 2: The improvement of leveraging a pre-trained autoencoding model to initialize the weights of the encoder in a seq2seq model is higher in the limited training data setting.

5.1.2 Field-level Evaluation. Table 5 shows our experiment results on the field-level. In terms of MRR@3, Rand2Rand achieves 0.555. This number means that the correct TAP is most likely located at half of the returned results before finding the relevant TAPs. We believe that this number is still acceptable for users. Similarly, Rand2Rand achieves 0.536 in terms of BLEU score. Leveraging RoBERTa and CodeBERT can improve the MRR@3 and BLEU score by 3.3%-3.9%. Also, note that these results are obtained by evaluating the model on Test+Field set that is not annotated by human. Therefore, there is a possibility that this test set contain more unclear descriptions than Noisy15.

Takeaway 3: Rand2Rand can achieve 0.555 and 0.536 in terms of MRR@3 and BLEU score. Instantiating the encoder with a pre-trained autoencoding model can improve the performance by 3.3%-3.6% in terms of MRR@3 and 3.5%-3.9% in terms of BLEU score.

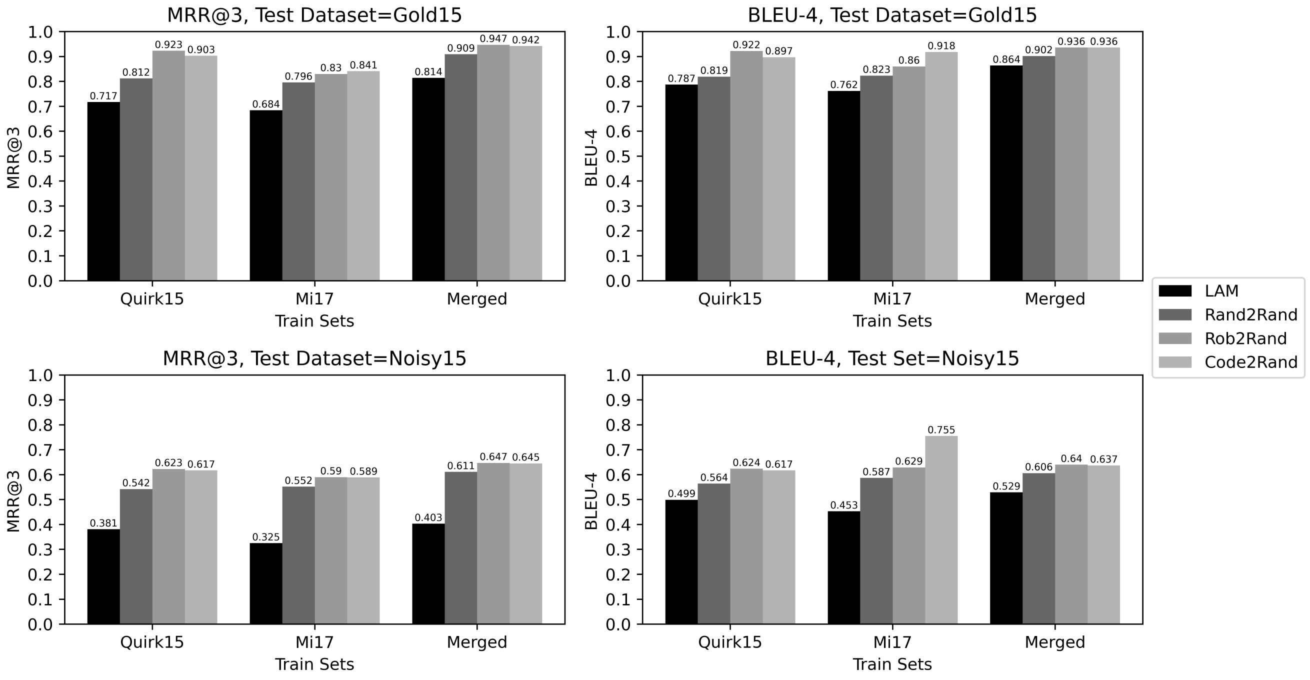

Figure 5: TAP generation results on function-level. Each model is trained on Quirk15, Mi17, Merged sets separately. The evaluation is performed on Gold15 (top) and Noisy15 (bottom) sets.

5.2 Does leveraging CodeBERT yield better performance improvement than RoBERTa?

Figure 5 shows that both Code2Rand and Robe2Rand yield competitive performance. We perform paired t-test to confirm whether the performance difference between Code2Rand and Rob2Rand is statistically significant. The differences on both Gold15 and Noisy15 are not statistically significant. The p-value on Gold15 and Noisy15 is 0.467 and 0.189 respectively. The possible reasons are the following. First, the corpora used in the pre-training phase may have different characteristics than TAP-related corpora. For instance, our datasets may contain domain-specific words such as channel or function names. Secondly, the objectives used to pre-train CodeBERT and RoBERTa are not designed for generation tasks. Although both models do not yield significant performance differences, the performance is still better than LAM as shown in Figure 5.

Takeaway 4: Both Rob2Rand and Code2Rand models perform better than LAM in terms of MRR@3 and BLEU. The two models do not have statistically significant performance differences.

6 DISCUSSION

We analyze some errors generated by RecipeGen to understand their causes. We also discuss implications and limitations of our study. Such discussions may be useful for future improvement work.

6.1 Error Analysis

6.1.1 Function-level. We pick the best-performing seq2seq models, i.e., Rob2Rand, and compare the accuracy of the individual components with LAM. We compute the accuracy by dividing the number of hits in the top-1 prediction by the total number of samples in the test set. In the function-level, Rob2Rand outperforms LAM in terms of individual accuracy in all of the components, as shown in Table 6. Such results corroborate our previous results in Section 4.

Table 6: Individual accuracy on Rob2Rand and LAM. Both models are trained on the Merged set. CH is channel, while FN is function.

Level	Test Set	Model	Accuracy (%)			
			Trigger		Action	
			CH	FN	CH	FN
Function	Gold	Rob2Rand	97.0	93.4	99.7	97.0
		LAM	85.6	72.6	82.3	74.8
	Noisy	Rob2Rand	95.1	88.5	96.1	94.4
		LAM	81.2	66.4	76.4	67.4
Field	Test+Field	Rob2Rand	79.0	68.1	73.0	65.3
		Code2Rand	78.5	67.5	73.7	66.0

We classify the possible errors into three cases as follows. ✓ indicates a correct prediction, while ✗ indicates an incorrect prediction.

Case 1: Channel ✓ and Function ✗. Two possible reasons for Case 1 error are 1) users mention only the channel names without describing the specific functionalities in the descriptions, and 2) the description for the functionality is too coarse-grained. For trigger, the number of such cases on Rob2Rand and LAM is 100 and 140 cases respectively out of 769 total samples. For action, Rob2Rand and LAM results 58 and 98 cases respectively out of 769 total samples. The example of Case 1 error is shown in Figure 6.

Case 2: Channel ✗ and Function ✓. The reason for Case 2 error is because users only describe the functionalities without mentioning the channel names. Multiple channels may have the

Send the github issues you're assigned to onenote

Ground Truth

Action channel	: OneNote
Action function	: OneNote.Create_a_page_from_a_link

Prediction

Action channel	: OneNote
Action function	: OneNote.OneNote.Create_a_page

Figure 6: Case 1 error; the action channel is correct and the action function is incorrect. The functionality description of action channel is not well-described and the action channel have similar functions with different granularity.

same or similar functionalities, as mentioned earlier. Rob2Rand does not yield any errors from this category. In contrast, LAM yields 26 and 29 cases for the trigger and action respectively out of 769 total samples. The example of Case 2 error is shown in Figure 7.

Send a notification if i don't get enough sleep

Ground Truth

Trigger channel	: Fitbit
Trigger function	: Fitbit.Sleep_duration_below

Prediction

Trigger channel	: UP_by_Jawbone
Trigger function	: Fitbit.Sleep_duration_below Notifications

Figure 7: Case 2 error; the trigger channel is incorrect and the trigger function is correct. Both Fitbit and UP by Jawbone are fitness tracker services that have sleep-related functionality.

Case 3: Channel ✗ and Function ✗. The reason for Case 3 error is because two channels may have very similar functionalities and the description does not specify specifically either the channel names or the functionalities. For trigger, the number of such cases on Rob2Rand and LAM is 111 and 118 cases respectively out of 769 total samples. For action, Rob2Rand and LAM results 136 and 152 cases respectively out of 769 total samples. The example of Case 3 error is shown in Figure 8.

Send me a new mobile notification for #ttc service advisories

Ground Truth

Action channel	: Pushover
Action function	: Pushover.Send_a_notification

Prediction

Action channel	: IF_Notifications
Action function	: IF_Notifications.Send_a_notification

Figure 8: Case 3 error; both the action channel and action function are incorrect. Both IF Notification and Pushover are channels for sending push-notifications to smartphones.

Overall, Rob2Rand has fewer errors than LAM in all error cases. Hence, our error analysis indicates that sequence learning is more suitable than the prior state-of-the-art which frames the problem as a classification task.

6.1.2 Field-level. Both Code2Rand and Rob2Rand performs competitively as shown in Table 6. An example of a correct prediction is shown in Figure 9. We can observe that RecipeGen can successfully

Sync "+diy projects/harry potter" ios album photos to evernote notebook "+diy projects/harry potter

Prediction

Trigger function	: iOS_Photos.New_photo_added_to_album
Trigger fields	: (Album name)
Action function	: Evernote.Create_image_note_from_URL
Action fields	: (Title, Image URL, Notebook, Tags)

Figure 9: A correct prediction in the field-level generation. This prediction is challenging because the function uses multiple field names.

generate both the trigger and action functions and identify their field names.

Figure 10 shows the example of the failed predictions in the field-level. Interestingly, the prediction is incorrect due to the unclear description instead of incorrectly predicting the trigger fields. Such an example illustrates that generating TAP with varying lengths is possible. One of the limitations of LAM is it assumes that the target sequence length is fixed. Therefore, the sequence is predicted using a set of fixed classifiers. Such a fact makes LAM does not scale if the target TAP is varying in length. In contrast, RecipeGen leverages sequence learning where the architecture of the model does not depend on the target sequence length. Consequently, RecipeGen is more scalable and can be easily extended to generate longer TAP sequences with varying length.

If nest cam detects motion, turn on a skylinknet lamp

Ground Truth

Trigger function	: Nest_Cam.New_motion_event
Trigger fields	: (Which device?)
Action function	: SkylinkNet.Turn_device_ON
Action fields	: ()

Prediction

Trigger function	: Nest_Cam.New_sound_or_motion_event
Trigger fields	: (Which device?)
Action function	: SkylinkNet.Turn_device_ON
Action fields	: ()

Figure 10: A failed prediction in the field-level generation. The trigger field is correct. However, the trigger function is incorrect as the intended functionality is to detect motion "only" but the predicted trigger functionality is detect "motion and sound".

6.2 Future Direction

There are a number of possible extensions of our work. The first extension is generating field values. Generating field values can be challenging as the description may not contain the necessary information to fill in the field values and the training data is too sparse to provide useful values. A possible solution is by framing the problem as a sequence completion problem and training an additional model using external knowledge source to fill in the values. Although it may not generate the actual intended values, the user still can use the generated field values as a suggestion. The second extension is proposing a technique to improve the adaptation of existing off-the-shelf pre-trained models. Our results indicate that leveraging a domain-specific pre-trained model (i.e., CodeBERT), does not yield significant performance difference compared to a general pre-trained model (i.e., RoBERTa). Such a fact suggests that there are opportunities to improve such adaptations using more sophisticated learning techniques.

6.3 Threats to Validity

Even though our idea may be generalizable to different TAP domains, there are still possible threats to external validity of our study and experiments. For example, we used the same dataset as the prior work in the TAP generation task [13, 15, 14], it does not cover all latest IFTTT data and it only contains TAPs with one trigger-action pair. For semantic correctness, we did not specifically validate whether the generated trigger/action components are valid. In real deployments, we envision that this issue can be resolved by validating each generated component against additional context information (e.g., the actual devices or services in users' environments) when performing beam search to discard invalid names from its generation candidates. There may also be threats to internal validity in the experiments and implementation that we may have made unintended bugs. We have released the replication package for others to check. Lastly, the ground truth and metrics that we use to evaluate our approach may not be suitable, inducing threats to construct validity. We also plan to devise new methods to improve the dataset and its ground truth as there could be more kinds of TAPs available and there could be more than one correct answer for each input description.

7 RELATED WORK

TAP has been scrutinized in various aspects, such as security [2, 37], privacy [38, 39, 40] testing [41], debugging [42], infrastructure deployment [43], and empirical study [44, 12, 10]. In this section, we discuss more thoroughly prior works that are closely related with our study, i.e., TAP generation and recommendation.

One-Shot Generation. Some approaches have been proposed to generate recipes automatically in one shot. Quirk et al. [13] framed the problem as a classification task and use logistic regression to generate recipes automatically. The authors treated the target recipe as a set of productions. They trained a binary classifier for each production by leveraging linguistic features from the source description (i.e., word unigrams and bigrams, and character trigrams) to decide whether production should appear in the output. However, this approach is not scalable because each production requires a different classifier. In contrast, the productions may grow because new channels appear or existing channels release new functions.

Beltalgy et al. [15] improved the prior approach by framing the problem as a structure prediction. Each modeling decision predicts one small component of the target structure, conditioned on the input and prior outputs. The authors built an ensemble of a logistic regression classifier and a Multi-Layer Perceptron (MLP). Although it can outperform the prior approach from Quirk et al. [13], the authors still ignore the source description's semantic because they treated the source features as bag-of-words.

The current state-of-the-art on one-shot recipe generation is Latent Attention Model (LAM) [14]. Similar to [13], the authors trained multi-class classifiers to predict the trigger and action instead of using a binary classifier. Moreover, the authors developed a new attention mechanism called latent attention. After computing, the standard attention [45, 46], LAM weights the attention using the latent weight of each token in the sentence. However, LAM ignores the relationship between trigger and action because the predictions are made disjointedly.

Interactive Generation. Huang et al. [47] introduced a framework called InstructableCrowd that allows users to compose applets by having a conversation with crowd-workers via a smartphone app. Chaurasia et al. [48] automate the prior approach from Huang et. al. [47] by replacing human crowd-workers with a conversational agent and casting the problem as a slot-filling task. Recently, Yao et al. [49] improved the conversational agent by leveraging hierarchical reinforcement learning to let the agent learn while interacting with users. However, such an approach is costly because the agent needs to interact with users in the training phase. Although simulation of human interaction can be an alternative, designing such a thing is a challenging task. Corno et. al. [50] also developed a conversational agent called HeyTap. HeyTAP leverages users' usages information and their specified profiles to make recommendations instead of only utilizing users' intentions as in the prior works [49, 48, 47].

TAP Recommendation. Corno et. al. introduced a tool that can give TAP recommendations to the users called RecRules [51]. RecRules represents channels and functions as semantic graph representation using manually crafted rules, given users' usage history. Subsequently, RecRules performs semantic reasoning using third-party software followed by collaborative filtering mapping. RecipeGen and RecRules is different because RecRules requires 1) TAP usages information of other users, and 2) mapping of channels and functions to the semantic graph representation. In contrast, RecipeGen only requires natural language description to make recommendations. Further, Corno et. al. then proposed an end-to-end framework based on RecRules called TapRec [11].

8 CONCLUSION

We have proposed RecipeGen to address the Trigger-Action Program (TAP) generation problem. RecipeGen formulate the problem as a sequence-to-sequence problem and leverages Transformer seq2seq architecture to translate an input description into a sequence of triggers and actions in the fine-grained field-level. We have evaluated RecipeGen on real-world datasets curated from the IFTTT platform against the prior state-of-the-art classification-based approach for TAP generation. Firstly, our results show that the seq2seq problem formulation generates more accurate TAPs containing channels and functions than the classification-based approach and can generate fine-grained fields that the prior approach cannot. Secondly, leveraging an autoencoding pre-trained model to warm-start the encoder improves the performance further. Thirdly, our results also indicate that a domain-specific and a general pre-trained models do not yield significant difference in terms of performance improvement.

Acknowledgments. This research is supported by the Ministry of Education, Singapore, under its Academic Research Fund Tier 2 (Award No.: MOE2019-T2-1-193). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the Ministry of Education, Singapore.

REFERENCES

[1] Sam S. Adams. 2008. The future of end user programming? In *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008, Companion Volume.* Wilhelm Schäfer, Matthew B. Dwyer, and Volker Gruhn, editors. ACM, 887–888. DOI: 10.1145/1370175.1370177. https://doi.org/10.1145/1370175.1370177.

[2] Lefan Zhang, Weijia He, Jesse Martinez, Noah Brackenbury, Shan Lu, and Blase Ur. 2019. Autotap: synthesizing and repairing trigger-action programs using ltl properties. In *2019 IEEE/ACM 41st international conference on software engineering (ICSE).* IEEE, 281–291.

[3] IFTTT. [n. d.] IFTTT helps every thing work better together. https://ifttt.com/. (Accessed on 08/29/2021). ().

[4] Olisto. [n. d.] Smart Connected Experiences - Olisto Brands. (Accessed on 08/03/2021).

[5] Integromat. [n. d.] Integromat - achieve more in less time with fewer people. (Accessed on 08/03/2021).

[6] Zapier. [n. d.] Zapier | The easiest way to automate your work. (Accessed on 08/03/2021).

[7] Microsoft Power Automate. [n. d.] Power automate | microsoft power platform. (Accessed on 08/03/2021).

[8] Home Assistant. [n. d.] Home Assistant. (Accessed on 08/03/2021).

[9] OpenHAB. [n. d.] OpenHAB. (Accessed on 08/03/2021).

[10] Xianghang Mi, Feng Qian, Ying Zhang, and XiaoFeng Wang. 2017. An empirical characterization of ifttt: ecosystem, usage, and performance. In *Proceedings of the 2017 Internet Measurement Conference*, 398–404.

[11] Fulvio Corno, Luigi De Russis, and Alberto Monge Roffarello. 2020. Taprec: supporting the composition of trigger-action rules through dynamic recommendations. In *Proceedings of the 25th International Conference on Intelligent User Interfaces*, 579–588.

[12] Blase Ur, Melwyn Pak Yong Ho, Stephen Brawner, Jiyun Lee, Sarah Mennicken, Noah Picard, Diane Schulze, and Michael L Littman. 2016. Trigger-action programming in the wild: an analysis of 200,000 ifttt recipes. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 3227–3231.

[13] Chris Quirk, Raymond Mooney, and Michel Galley. 2015. Language to code: learning semantic parsers for if-this-then-that recipes. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 878–888.

[14] Chang Liu, Xinyun Chen, Eui Chul Shin, Mingcheng Chen, and Dawn Song. 2016. Latent attention for if-then program synthesis. *Advances in Neural Information Processing Systems*, 29, 4574–4582.

[15] Islam Beltagy and Chris Quirk. 2016. Improved semantic parsers for if-then statements. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 726–736.

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, 5998–6008.

[17] Fulvio Corno, Luigi De Russis, and Alberto Monge Roffarello. 2019. Eudoptimizer: assisting end users in composing if-then rules through optimization. *IEEE Access*, 7, 37950–37960. DOI: 10.1109/ACCESS.2019.2905619.

[18] Andrew M Dai and Quoc V Le. 2015. Semi-supervised sequence learning. *Advances in neural information processing systems*, 28, 3079–3087.

[19] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

[20] Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*.

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

[22] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: a pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*.

[23] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1, 8, 9.

[24] Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. 2021. Codet5: identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *arXiv preprint arXiv:2109.00859*.

[25] Clara Meister, Ryan Cotterell, and Tim Vieira. 2020. If beam search is the answer, what was the question? In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP).* Association for Computational Linguistics, Online, (November 2020), 2173–2185. DOI: 10.18653/v1/2020.emnlp-main.170. https://aclanthology.org/2020.emnlp-main.170.

[26] Xin Zhou, DongGyun Han, and David Lo. 2021. Assessing generalizability of codebert. In *IEEE International Conference on Software Maintenance and Evolution, ICSME 2021, Luxembourg, September 27 - October 1, 2021.* IEEE, 425–436. DOI: 10.1109/ICSME52107.2021.00044. https://doi.org/10.1109/ICSME52107.2021.00044.

[27] Ehsan Mashhadi and Hadi Hemmati. 2021. Applying codebert for automated program repair of java simple bugs. In *18th IEEE/ACM International Conference on Mining Software Repositories, MSR 2021, Madrid, Spain, May 17-19, 2021.* IEEE, 505–509. DOI: 10.1109/MSR52588.2021.00063. https://doi.org/10.1109/MSR52588.2021.00063.

[28] Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S Weld, Luke Zettlemoyer, and Omer Levy. 2020. Spanbert: improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8, 64–77.

[29] Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. 2018. Deep code search. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE).* IEEE, 933–944.

[30] Jing Kai Siow, Cuiyun Gao, Lingling Fan, Sen Chen, and Yang Liu. 2020. Core: automating review recommendation for code changes. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER).* IEEE, 284–295.

[31] Fei Lv, Hongyu Zhang, Jian-guang Lou, Shaowei Wang, Dongmei Zhang, and Jianjun Zhao. 2015. Codehow: effective code search based on api understanding and extended boolean model (e). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE).* IEEE, 260–270.

[32] Xin Ye, Razvan Bunescu, and Chang Liu. 2014. Learning to rank relevant files for bug reports using domain knowledge. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 689–699.

[33] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 311–318.

[34] Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim. 2016. Deep api learning. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 631–642.

[35] Zeyu Sun, Qihao Zhu, Lili Mou, Yingfei Xiong, Ge Li, and Lu Zhang. 2019. A grammar-based structural cnn decoder for code generation. In *Proceedings of the AAAI conference on artificial intelligence* number 01. Volume 33, 7055–7062.

[36] Zeyu Sun, Qihao Zhu, Yingfei Xiong, Yican Sun, Lili Mou, and Lu Zhang. 2020. Treegen: a tree-based transformer architecture for code generation. In *Proceedings of the AAAI Conference on Artificial Intelligence* number 05. Volume 34, 8984–8991.

[37] Sunil Manandhar, Kevin Moran, Kaushal Kafle, Ruhao Tang, Denys Poshyvanyk, and Adwait Nadkarni. 2020. Towards a natural perspective of smart homes for practical security and safety analyses. In *2020 IEEE Symposium on Security and Privacy (SP).* IEEE, 482–499.

[38] Milijana Surbatovich, Jassim Aljuraidan, Lujo Bauer, Anupam Das, and Limin Jia. 2017. Some recipes can do more than spoil your appetite: analyzing the security and privacy risks of ifttt recipes. In *Proceedings of the 26th International Conference on World Wide Web*, 1501–1510.

[39] Rixin Xu, Qiang Zeng, Liehuang Zhu, Haotian Chi, Xiaojiang Du, and Mohsen Guizani. 2019. Privacy leakage in smart homes and its mitigation: ifttt as a case study. *IEEE Access*, 7, 63457–63471.

[40] Camille Cobb, Milijana Surbatovich, Anna Kawakami, Mahmood Sharif, Lujo Bauer, Anupam Das, and Limin Jia. 2020. How risky are real users' IFTTT applets? In *Sixteenth Symposium on Usable Privacy and Security (SOUPS) 2020)*, 505–529.

[41] Kulani Mahadewa, Yanjun Zhang, Guangdong Bai, Lei Bu, Zhiqiang Zuo, Dileepa Fernando, Zhenkai Liang, and Jin Song Dong. 2021. Identifying privacy weaknesses from multi-party trigger-action integration platforms. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2–15.

[42] Fulvio Corno, Luigi De Russis, and Alberto Monge Roffarello. 2019. Empowering end users in debugging trigger-action rules. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 1–13.

[43] Lei Liu, Mehdi Bahrami, and Wei-Peng Chen. 2020. Automatic generation of ifttt mashup infrastructures. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE).* IEEE, 1179–1183.

[44] Blase Ur, Elyse McManus, Melwyn Pak Yong Ho, and Michael L Littman. 2014. Practical trigger-action programming in the smart home. In *Proceedings of the SIGCHI conference on human factors in computing systems*, 803–812.

[45] Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. 2015. English (US). In 3rd International Conference on Learning Representations, ICLR 2015 ; Conference date: 07-05-2015 Through 09-05-2015. (January 2015).

[46] Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the*

2015 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, Lisbon, Portugal, (September 2015), 1412–1421. DOI: 10.18653/v1/D15-1166. https://aclanthology.org/D15-1166.

[47] Ting-Hao Kenneth Huang, Amos Azaria, and Jeffrey P Bigham. 2016. Instructablecrowd: creating if-then rules via conversations with the crowd. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, 1555–1562.

[48] Shobhit Chaurasia and Raymond Mooney. 2017. Dialog for language to code. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, 175–180.

[49] Ziyu Yao, Xiujun Li, Jianfeng Gao, Brian Sadler, and Huan Sun. 2019. Interactive semantic parsing for if-then recipes via hierarchical reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence* number 01. Volume 33, 2547–2554.

[50] Fulvio Corno, Luigi De Russis, and Alberto Monge Roffarello. 2021. From users' intentions to if-then rules in the internet of things. *ACM Transactions on Information Systems (TOIS)*, 39, 4, 1–33.

[51] Fulvio Corno, Luigi De Russis, and Alberto Monge Roffarello. 2019. Recrules: recommending if-then rules for end-user development. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10, 5, 1–27.