RecipeGen++: An Automated Trigger Action Programs Generator

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

ABSTRACT

Trigger Action Programs (TAPs) are event-driven rules that allow users to automate smart-devices and internet services. Users can write TAPs by specifying triggers and actions from a set of predefined channels and functions. Despite its simplicity, composing TAPs can still be challenging for users due to the enormous search space of available triggers and actions. The growing popularity of TAPs is followed by the increasing number of supported devices and services, resulting in a huge number of possible combinations between triggers and actions. Motivated by such a fact, we improve our prior work and propose *RecipeGen++*, a deep-learning-based approach that leverages Transformer seq2seq (sequence-to-sequence) architecture to generate TAPs given natural language descriptions. *RecipeGen++* can generate TAPs in the *Interactive*, *One-Click*, or *Functionality Discovery* modes. In the Interactive mode, users can provide feedback to guide the prediction of a trigger or action component. In contrast, the One-Click mode allows users to generate all TAP components directly. Additionally, *RecipeGen++* also enables users to discover functionalities at the channel level through the Functionality Discovery mode. We have evaluated *RecipeGen++* on real-world datasets in all modes. Our results demonstrate that *RecipeGen++* can outperform the baseline by 2.2%-16.2% in the gold-standard benchmark and 5%-29.2% in the noisy benchmark.
Demo: https://4ek5.short.gy/RecipeGen
Tool: https://huggingface.co/spaces/imamnurby/RecipeGen
GitHub: https://github.com/imamnurby/RecipeGen

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, Diyanah Binte Abdul Jamal, Lingxiao Jiang, David Lo. 2022. *RecipeGen++*: An Automated Trigger Action Programs Generator. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22), November 14–18, 2022, Singapore, Singapore.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3540250.3558913

ESEC/FSE '22, November 14–18, 2022, Singapore, Singapore

ACM ISBN 978-1-4503-9413-0/22/11…$15.00
https://doi.org/10.1145/3540250.3558913

1 INTRODUCTION

Trigger Action Programs (simply TAPs) are event-driven rules used to automate smart devices and online services. The rules are typically written in the form of "IF a *trigger* is satisfied, then execute an *action*". The *trigger* and *action* correspond to API calls provided by service providers. TAPs have been applied in various use cases, such as home automation [16, 20], business [12, 17], and data management [2–4]. In this work, we are interested in generating TAPs automatically using natural language descriptions.

Description from a User
Saves any photo tagged with me in it on fb to Dropbox
TAP
Trigger : Facebook.You_Are_Tagged_In_Photo()
Action : Dropbox.Add_File_From_Url(URL, Name, Path)

Figure 1: A simplified TAP and its description.

Figure 1 shows an example of a TAP. To compose a TAP, users usually specify a trigger and an action sequentially. For each trigger and action, the user needs to select three components, i.e., a channel, a function, and a set of fields. A channel is an entity that provides services. A function is an API that represents a particular service. For instance, in Figure 1, the trigger channel is "Facebook" and the trigger function is "You_Are_Tagged_In_Photo". Fields are additional parameters needed to control how a function works. A field is composed of a field name and a field value. Fields can be either requisite or optional. In Figure 1, the action function has several field names, i.e., "URL", "Name", and "Path". The field values correspond to the actual values given by users.

Writing TAPs can be challenging due to the enormous search space when choosing trigger and action components. To exemplify, in IFTTT, there are 408 distinct channels resulting in 468,930 possible combinations of trigger and action functions in 2017 [16]. The number of channels becomes 1,386 in 2021 [11], causing such possible combinations to increase further. Hence, there is a need for a tool that can help end-users to generate TAPs from such an enormous search space.

Such a fact motivates a number of researchers to propose approaches to generate TAPs using natural language descriptions. The existing approaches adopt classification-based techniques to generate TAPs [1, 13, 18]. These approaches employ a number of *distinct* classifiers to predict each trigger and action component individually as illustrated in Figure 2 (a). Consequently, such approaches cannot perform well on ambiguous descriptions because it cannot leverage the implicit relationship between each predicted component. We address this drawback in our recent work by proposing RecipeGen [23].

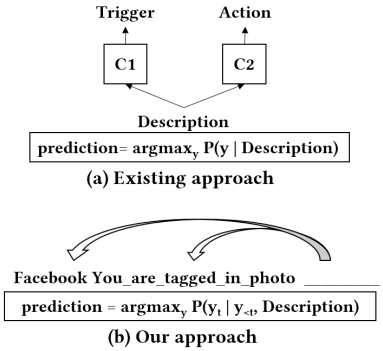

Figure 2: (a) A classification model predicts each component in isolation given a description, while (b) RecipeGen considers both the prior predicted components and the description when generating a prediction.

RecipeGen leverages Transformer seq2seq architecture [21] to translate a description to a sequence of trigger and action components. As illustrated in Figure 2 (b), RecipeGen considers the prior predicted components as additional information besides the description to generate a TAP component. This mechanism allows RecipeGen to leverage the implicit relationship between each predicted component. Moreover, RecipeGen adapts autoencoding pre-trained models to warm-start the encoder in the seq2seq model to boost the generation performance.

Although the aforementioned works demonstrate great potential, they only serve as a prototype to illustrate the viability of a research idea rather than to help users compose TAPs in the real world. Novice users may face difficulties using existing tools because they often still need to set up the tools and their environments. Moreover, existing tools also force users to interact through a command-line interface. The lack of an easy-to-use tool to help users compose TAPs motivates us to improve our original tool and propose *RecipeGen++*. First, we add a graphical user interface in *RecipeGen++* to interact with end-users and additional modules to prepare the input and output (see Figure 4). Second, we add the *Interactive* generation mode in *RecipeGen++* besides the existing *One-Click* mode. The Interactive mode guides users to compose TAPs step by step, resulting in users having more flexibility when composing such TAPs. Third, we add the *Functionality Discovery* mode to let users discover more functionalities by generating TAPs at the channel level. This mode is motivated by Corno et. al. [7] who found that: users may not realize that a more suitable function exists for their use case as generating TAPs at the function level often yield results that do not capture such a function.

We have evaluated *RecipeGen++* on real-world datasets in each mode. Our results demonstrate that *RecipeGen++* can yield better performance against the baseline by 2.2%-16.2% in the gold-standard benchmark and 5%-29.2% in the noisy benchmark. Our work makes the following contributions:

- We propose *RecipeGen++*, the first usable tool that leverages the Transformer seq2seq architecture to generate TAPs using functionality descriptions. *RecipeGen++* frames the problem as a sequence learning problem and leverages a pre-trained model to warm-start the encoder.

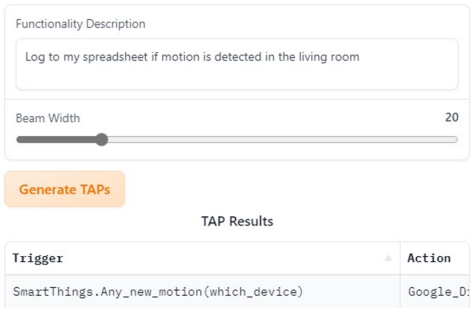

Figure 3: User Interface of *RecipeGen++*.

- We deploy *RecipeGen++* such that the community can use our tool[1] directly, without additional setup or installation steps. Moreover, we also release the codebase[2] to train the model and replicate the results.

The rest of this paper is organized as follows. Section 2 explains the usage of our tool. Section 3 explains the technical details of *RecipeGen++*. Section 4 describes the evaluation setting and presents the results. Section 5 discusses related works. Finally, Section 6 concludes our work.

2 TOOL USAGE

RecipeGen++ has three working modes, i.e., *Interactive*, *One-Click*, and *Functionality Discovery*. First, users who want to have more flexibility when composing TAPs can use the Interactive mode. This mode guides the users using a step-by-step wizard. Users can select a component (e.g., trigger channel) from a set of predicted TAP components at each step, then use the selected component as feedback to the model to guide the next TAP component prediction (e.g., trigger function). In contrast, users who prefer to see the TAP candidates quickly for a given description can leverage the One-Click mode. This mode allows users to generate all TAP components using a one-click button press. Users who are not sure yet which channels or functions to use can explore the available channels and functions through the Functionality Discovery mode. This mode enables users to find relevant channels based on the given description and allows them to inspect their functions *exhaustively*.

Figure 3 shows the user interface of *RecipeGen++*. Users can describe the intended TAP in the *Functionality Description* text box and tune the number of TAP candidates using the *Beam Width* slider. Afterward, users can generate TAP candidates by pressing the *Generate TAPs* button. The generated candidates are shown in the table at the bottom. Note that the table can be scrolled horizontally. The TAP candidates are ordered based on the probability prediction; a result with a high ranking is more relevant according to the model than those that are lower.

3 *RECIPEGEN++*

This section explains the workflow of *RecipeGen++* that consists of two stages: training and deployment. Figure 4 shows the modules involved in each stage. In the training stage, the Initializer

[1] https://huggingface.co/spaces/imamnurby/RecipeGen
[2] https://github.com/imamnurby/RecipeGen

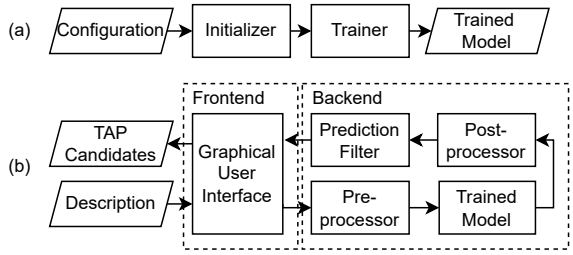

Figure 4: The workflow of *RecipeGen++* consists of two stages: Training and Deployment. The modules involved in the former and latter stages are indicated by (a) and (b), respectively.

receives a configuration file as the input to instantiate the model. Subsequently, Trainer takes the model as the input and trains it to translate descriptions to TAPs. In the Deployment stage, the Frontend receives inputs from users and displays outputs from the model to users. Moreover, Backend modules have a role to perform preprocessing to user inputs, generate TAP candidates, and perform postprocessing to model outputs.

3.1 Training

First, we explain the general training pipeline that involves Initializer and Trainer. Second, we explain the technique to augment the training data to train a model for the Interactive mode.

3.1.1 Training Pipeline. Given a configuration file, Initializer instantiates the model using the Transformer seq2seq architecture. Subsequently, Initializer warm-starts the model encoder weights using a pre-trained model. The reason is that: a pre-trained model has been trained using a huge amount of data from English corpora, and TAP descriptions have a similar nature to such data (see Figure 1). Consequently, the learning burden can be reduced if the encoder can have such knowledge at the beginning of its training. Then, the Trainer trains the model to translate functionality descriptions to TAPs. The Trainer input is ⟨*description, TAP*⟩ pairs. First, some special tokens are inserted into the descriptions and TAPs. Second, a tokenizer converts the descriptions and TAPs into a sequence of subword tokens. For example, "<s> Saves any photo to dropbox </s>" is tokenized into "<s>", "Save","any", "photo", "to", "drop", "box", "</s>". The motivation for tokenizing both descriptions and TAPs to subword level is to let the model exploit the *subword* similarity between the descriptions and TAPs. Third, the tokenizer converts each subword token to the corresponding token ids $x_1, x_2, ..., x_n$. Fourth, the encoder transforms it into a sequence of context vectors $c_1, c_2, ..., c_n$. Finally, the decoder takes these context vectors and the sequence of token ids from TAPs $y_1, y_2, ..., y_n$ as the input to generate the predictions $p_2, p_3, ..., p_{n-1}$.

3.1.2 Data Augmentation for Interactive Model. The idea of the Interactive model is to predict TAP components in several steps, where each step corresponds to one TAP component prediction. To train the model to generate TAPs in such a way, the descriptions in the training instances should contain information about which component to generate and components that have been selected by users. Hence, we apply Algorithm 1 to augment the original training pairs with such information. Algorithm 1 converts descriptions in the original pairs to *prefix + description + feedback* (line 7), where *prefix* is used to indicate which component to generate and *feedback* corresponds to synthetic user feedback. The feedback is obtained by concatenating the TAP components before the current target component (line 5).

Algorithm 1: Generating training instances for interactive generation

Input : *description*: a description of a TAP
components: an ordered-list of TAP components
prefixes: a dictionary that maps a TAP component to a prefix
Output : *outputs*: a list of training instances

```
1 outputs = [];
2 n_components = getLength(components);
3 for idx in range(n_components) do
4     target_component = components[idx];
5     feedback = convertToStr(components[:idx]);
6     prefix = prefixes.get(target_component);
7     final_description = prefix + description + feedback;
8     outputs.append((final_description, target_component))
9 end for
```

3.2 Deployment

The deployment stage involves a Graphical User Interface (GUI) module and a number of Backend modules. The input of the deployment stage is a description and the output is a ranked list of TAP candidates.

3.2.1 Graphical User Interface (GUI). The GUI module allows users to interact with *RecipeGen++* easily: users can provide functionality descriptions through a text box, generate predictions using a button, and inspect the generated predictions through a graphical interface.

3.2.2 Backend Modules. The Backend consists of several modules. The Preprocessor transforms the description into a sequence of token ids. The Trained Model then takes these token ids as the input to generate the TAP candidates. The Postprocessor processes the TAP candidates by removing the special tokens and converting them into a compatible representation for the GUI module. As we mentioned previously, the model generates the prediction at the subword level. Consequently, the model may generate invalid TAP components because a set of predicted subwords does not form a valid component (e.g., non-existent channel or function). Hence, the Prediction Filter verifies each generated component to avoid such a case. If the Prediction Filter finds invalid components, then such components will be discarded.

4 EVALUATION AND RESULTS

4.1 Experimental Setting

4.1.1 Models. We train *RecipeGen++* and leverage RoBERTa [14] to warm-start the encoder. For the baseline, we use LAM [13] which is the best classification-based approach for TAP generation. We

do not specifically compare with RecipeGen [23] because it is the same as *RecipeGen++* that works in One-Click mode.

4.1.2 Datasets. We use the IFTTT datasets from Quirk et. al. [18] and Mi et. al. [16] to train and evaluate each model. Each instance in the datasets is a ⟨*description, TAP*⟩ pair, where each TAP is a sequence of a trigger channel, trigger function, action channel, and action function. There are two test sets: Gold and Noisy. Both test sets are human-annotated. The annotation process is done by labeling each description with the TAP implementation. The Gold set contains the instances with at least three annotator agreements. The instances with less than three agreements belong to the Noisy set.

4.1.3 Scenario. We evaluate each model in three different settings: One-Click, Interactive, and Functionality Discovery. We leverage the original pairs from the test sets in the One-Click setting. We omit the function components from the original TAPs in the Functionality Discovery setting. We apply Algorithm 1 to convert the original pairs to ⟨*prefix + description + feedback, component*⟩ pairs in the Interactive setting.

4.1.4 Metrics. We leverage Mean Reciprocal Rank (MRR) as the evaluation metrics [9, 15, 19, 22]. MRR@k is computed using Equation 1:

$$MRR@k = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i} \quad (1)$$

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 indicates the average number of predictions that a user needs to investigate to find the correct one.

Table 1: MRR@3 results on Gold and Noisy set at different setting.

Setting	Model	MRR@3	
		Gold	Noisy
Functionality Discovery	*RecipeGen++*	**0.990**	**0.846**
	LAM	0.891	0.672
One-Click	*RecipeGen++*	**0.967**	**0.752**
	LAM	0.805	0.460
Interactive	*RecipeGen++*	**0.987**	**0.930**
	LAM	0.965	0.880

4.2 Results

Table 1 shows the MRR@3 results on the Gold and Noisy sets in three different settings. First, *RecipeGen++* outperforms LAM by 9.9% and 17.5% on Gold and Noisy sets in the Functionality Discovery setting. Such results indicate that *RecipeGen++* can have high accuracy in finding the relevant channels using the given description. The improvements become 16.2% and 29.2%, respectively, when the models generate channels along with functions in the One-Click setting. Such results indicate that *RecipeGen++* can rank TAPs with relevant functions better than LAM. Moreover, the improvements on the Gold and Noisy sets in the Interactive setting are 2.2% and 5%, respectively. The improvements are smaller than the other two settings because the target TAPs consist of only one component, making the prediction easier. However, the significant improvement on the Noisy set against other modes may indicate that user feedback can be useful in resolving ambiguous descriptions. Overall, our results in Table 1 demonstrate that *RecipeGen++* can outperform LAM in all evaluation settings.

5 RELATED WORK

Several approaches have been proposed to help users compose TAPs. Quirk et al. [18] trained classifiers using linguistic features such as word n-grams. Beltalgy et al. [1] improved the previous approach by combining a logistic regression classifier with a Multi-Layer Perceptron (MLP). Both approaches [1, 18] still ignore the description's semantics because they treat the description as bag-of-words. This limitation is addressed by LAM [13] using LSTM-based classifiers. However, LAM still performs poorly when the description is ambiguous due to formulating the problem as classification. *RecipeGen++*, which is our proposed approach, addresses this limitation by framing the problem as sequence learning. Some studies to generate TAPs interactively have also been carried out. Huang et al. [10] introduced a framework called InstructableCrowd that allows users to compose TAPs by having a conversation with crowd-workers via a smartphone app. Chaurasia et al. [5] automate the previous approach by replacing the crowd-workers with a conversational agent built using LAM. Although it can scale better, the conversational agent still has the same drawback as LAM because it leverages LAM as the core. Recently, Corno et. al. also developed several approaches to recommend TAPs [6–8]. However, these approaches require additional information such as usage history, user preference, and semantic graph representation. In contrast, *RecipeGen++* only requires users to input functionality descriptions to generate TAPs.

6 CONCLUSION AND FUTURE WORK

This work presents *RecipeGen++*, a tool to help users compose TAPs using functionality descriptions of the intended TAPs. *RecipeGen++* frames the problem as sequence learning to allow the model to leverage the implicit relationship between each predicted component. Moreover, *RecipeGen++* also warm-starts the encoder using a pre-trained model to boost the generation performance. *RecipeGen++* can assist users in three different working modes: Interactive, One-Click, and Functionality Discovery. Our evaluation results show that *RecipeGen++* can outperform the baseline with significant margins in all working modes.

In the future, we plan to extend our tool to generate field values instead of only field names. Generating field values can be challenging as the description may not contain the necessary information to fill in the field values. Moreover, we also plan to adapt *RecipeGen++* to generate a TAP with multiple triggers and actions.

ACKNOWLEDGMENTS

This research / project 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] 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.

[2] Ryan Chard, Kyle Chard, Jason Alt, Dilworth Y Parkinson, Steve Tuecke, and Ian Foster. 2017. Ripple: Home automation for research data management. In *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE, 389–394.

[3] Ryan Chard, Kyle Chard, Steve Tuecke, and Ian Foster. 2017. Software defined cyberinfrastructure for data management. In *2017 IEEE 13th International Conference on e-Science (e-Science)*. IEEE, 456–457.

[4] Ryan Chard, Rafael Vescovi, Ming Du, Hanyu Li, Kyle Chard, Steve Tuecke, Narayanan Kasthuri, and Ian Foster. 2018. High-throughput neuroanatomy and trigger-action programming: a case study in research automation. In *Proceedings of the 1st International Workshop on Autonomous Infrastructure for Science*. 1–7.

[5] 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.

[6] 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 (2019), 1–27.

[7] 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.

[8] 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 (2021), 1–33.

[9] 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.

[10] 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.

[11] IFTTT. n.d. Explore - IFTTT. Retrieved June 9, 2022 from https://ifttt.com/

[12] Integromat. n.d. Integromat - Achieve more in less time with fewer people. Retrieved June 9, 2022 from https://www.integromat.com/en

[13] 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 (2016).

[14] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[15] 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.

[16] 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.

[17] Microsoft Power Automate. n.d. Power Automate | Microsoft Power Platform. Retrieved June 9, 2022 from https://powerautomate.microsoft.com/en-us/

[18] 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.

[19] 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.

[20] 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.

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[22] 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.

[23] Imam Nur Bani Yusuf, Lingxiao Jiang, and David Lo. 2022. Accurate Generation of Trigger-Action Programs with Domain-Adapted Sequence-to-Sequence Learning. In *Proceedings of the 30th International Conference on Program Comprehension*.