Revisiting Learning-based Commit Message Generation

***Abstract*—Commit messages summarize code changes and help developers understand the intention. To alleviate human efforts in writing commit messages, researchers have proposed various automated commit message generation techniques, among which learning-based techniques have achieved great success in recent years. However, existing evaluation on learning-based commit message generation relies on the automatic metrics (e.g., BLEU) widely used in natural language processing (NLP) tasks, which are aggregated scores calculated based on the similarity between generated commit messages and the ground truth. Therefore, it remains unclear what generated commit messages look like and what kind of commit messages could be precisely generated by existing learning-based techniques.**

To fill this knowledge gap, this work performs the first study to systematically investigate the detailed commit messages generated by learning-based techniques. In particular, we first investigate the *frequent patterns* of the commit messages generated by state-of-the-art learning-based techniques. *Surprisingly, we find the majority (˜90%) of their generated commit messages belong to simple patterns (i.e., addition/removal/fix/avoidance patterns).* To further explore the reasons, we then study the impact of datasets, input representations, and model components. We surprisingly find that existing learning-based techniques have competitive performance even when the inputs are only represented by change marks (i.e., "+"/"-"/" "). It indicates that existing learning-based techniques poorly utilize syntax and semantics in the code while mostly focusing on change marks, which could be the major reason for generating so many pattern-matching commit messages. We also find that the pattern ratio in the training set might also positively affect the pattern ratio of generated commit messages; and model components might have different impact on the pattern ratio.

I. Introduction

When developers submit code changes in version control systems, they are supposed to attach a *commit message* to summarize their submitted code changes. Commit messages are written in natural language and indicate *what* is changed and *why* changes happen. Commit messages are of vital importance for software maintenance, because developers often read commit messages to understand the implementation rationale and code semantics. In addition, many important software engineering tasks such as automated release-note generation [1], [2] and identification of bug-introducing commits [3], [4] rely on high-quality commit messages.

However, it is laborious to write high-quality commit messages to summarize code changes. With the rapid iteration of the software, writing commit messages burdens developers. Thus, there are often low-quality commit messages or empty commit messages. Prior work shows that around 14% of commit messages in more than 23K open-source Java projects are completely empty [5].

Given the importance of commit messages and the required manual effort in writing them, researchers have proposed various automatic commit message generation techniques, including rule-based [6]–[8], information retrieval-based [9]–[11] and learning-based [12]–[16] techniques. With the development of deep learning models, especially in NLP domain, learning-based commit message generation techniques have achieved substantial improvements. Although there are many learning-based techniques, most of them are evaluated with NLP metrics (e.g., BLEU). Since these NLP metrics can only measure the overall performance by aggregated scores, it still remains unclear what the generated commit messages look like and what kind of commit messages could be precisely generated by existing learning-based techniques.

In this work, we perform the first study to systematically investigate the detailed commit messages generated by learning-based techniques. In particular, we first distill the *frequent patterns* [1] of the commit messages generated by state-of-the-art learning-based techniques in a semi-auto way. We leverage sequential pattern mining algorithm to automatically mine raw frequent patterns in the generated commit messages, based on which we further manually summarize frequent patterns. Typically, the frequent patterns are often of simple format, e.g., the format of Addition Pattern is ***"Add [missing] ... [for|to ...]"***, and a real case is "*Add missing null check to shouldBackup method*". Compared with NLP metrics, we believe that patterns could reflect more details (e.g., what is changed and why) of the generated commit messages. Therefore, in the first research question, we statistically investigate what generated commit messages look like via the lense of patterns.

- **RQ1: What kind of commit messages could be generated by existing learning-based techniques?**

The main finding for RQ1 is as follows.

Findings of RQ1: The majority (i.e., ˜90%) of commit messages generated by learning-based techniques belong to simple patterns, and the ratio is much higher than that in ground truth (˜50%). Interestingly, although achieving not bad performance regarding NLP metrics, the majority of commit messages generated by learning-based techniques belong to simple patterns, such as removal (e.g., "Remove unused ...") and fix patterns (e.g., "Add ... to ..."). In addition, we also find that learning-based techniques have a bias towards generating commit messages of short length (4.5 tokens on

[1] A frequent pattern is a subsequence which frequently appears among a given corpus.

average), indicating that they still have limited performance on generating flexible/complex commit messages.

To further figure out the reasons for such a high ratio of pattern-matching commit messages, we then systematically study the impact of datasets, input representations, and model components on the effectiveness of existing learning-based techniques by the following research questions. Different from existing work, which evaluates commit message generation techniques with NLP metrics, our focus on the effectiveness is mainly related to the pattern of generated commit messages.

- **RQ2/3/4: How do datasets, input representations, and model components affect the effectiveness of learning-based commit message generation techniques?**

The main findings of each RQ are as follows.

Findings of RQ2: The ratio of commit messages fitting with patterns in the generated commit messages is positively associated with the ratio in the training set. The relatively high ratio of patterns in the training set might be one reason for the exorbitantly high ratio in the generated commit messages. Besides, even when increasing the ratio of non-patterns in the training set, the quality of generated non-pattern commit messages remains broadly unchanged, indicating that non-pattern data are challenging for models to learn.

Findings of RQ3: Existing learning-based techniques have competitive performance even when the inputs are only represented by change marks (i.e., "+"/"-"/" "). Change marks refer to the character leading each line in the code changes, including "+"/ "-"/" ", which means that in the new version this line is added/removed/unchanged compared to the old version. This finding implies that existing techniques mainly utilize marks for commit message generation and fail to capture the syntax and semantics of input code, which could be the major reason why they could only generate commit message of simple patterns.

Findings of RQ4: Model components have different influence on the ratio of pattern-matching commit messages. Moreover, models pay more attention to tokens with changed marks (i.e., "+" and "-") than unchanged marks (i.e., " ").

In summary, this paper makes the following contributions:

(1) A novel perspective on *pattern* to evaluate generated commit messages. Compared to NLP metrics, patterns could show the detailed distribution and structures of messages.

(2) A comprehensive evaluation on the commit message generation techniques regarding both newly-proposed pattern metrics and traditional NLP metrics.

(3) A deep exploration on the factors relevant to the high ratio of patterns, including datasets, input representations, and model components.

II. Background and Related Work

A. Commit Message Generation Techniques

Existing commit message generation techniques include rule-based [6]–[8], information retrieval-based [9]–[11] and learning-based [12]–[16] techniques. *Rule-based techniques* use pre-defined rules , whose generated commit messages are often verbose and become less effective when code changes do not fit any rules. *Information retrieval-based techniques* select the commit messages whose code changes are similar to the query one from the database, and would be less effective when there is no similar code changes in the database. With the recent development of deep learning, *learning-based techniques* have achieved great improvement. Learning-based techniques regard commit message generation as a translation task and leverage models with encoder-decoder architectures. TABLE I shows five state-of-the-art learning-based techniques, which are also the studied techniques in this study. The second column presents the encoder-decoder architecture adopted in each technique, and the last three columns present whether each technique incorporates attention, copy, and anonymization mechanisms. In particular, (1) *attention mechanism* [17], [18] makes the decoder focus on most valuable input tokens when generating each output token; (2) *copy mechanism* [19], [20] copies tokens from code changes to address the out-of-vocabulary (OOV) problem [20], [21]; (3) *anonymization mechanism* [14] replaces identifiers in code changes with placeholders, which can reduce the vocabulary size and solve the OOV problem when new identifiers appear.

TABLE I: The state-of-the-art learning-based techniques.

Techniques	Encoder-Decoder Architecture	Attention Mechanism	Copy Mechanism	Anonymization Mechanism
NMT [12]	GRU-GRU	✔	✖	✖
PtrGN [13]	GRU-GRU	✔	✔	✖
CODIS [14]	GRU-GRU	✔	✔	✔
CoreGen [15]	Transformer-Transformer	✔	✖	✖
FIRA [16]	GNN-Transformer	✔	✔	✔

B. Existing Metrics on Commit Message Generation

Existing techniques [12]–[16] evaluate the generated commit messages with the NLP metrics BLEU, ROUGE-L, and METEOR, which actually calculate the similarity between generated commit messages and the ground truth.

BLEU [22] measures the precision of the generated sequence, which is the average of the modified n-gram precision (i.e., 1-gram, 2-grams, 3-grams and 4-grams for BLEU-4). The modified n-gram precision refers to the ratio of matched n-grams to the n-grams in the generated sequence.

ROUGE-L [23] is a F-score based on the longest common sub-sequences (LCS) between two sequences. A longer LCS indicates a higher similarity between two sentences.

METEOR [24] is a F-score of 1-gram precision and 1-gram recall between the generated sequence and the ground truth, with a penalty when the order of matched tokens is wrong.

Although widely used, all these metrics are aggregated scores, and thus cannot reflect the detailed distribution and structures of generated commit messages. Therefore, besides the metrics, in this work, ***we propose a novel perspective, i.e., pattern, to evaluate generated commit messages***.

C. Existing Studies on Commit Message Generation

There are some existing studies on commit message generation. Tao et al. [25] evaluate existing commit message

generation techniques with different BLEU metrics and find different results between different BLEU variants. Tian et al. [26] define a standard for "good" commit messages by manually investigating real-world commit messages. Different from existing studies, our work proposes a novel perspective, i.e., pattern, to study the details and distribution of commit messages generated by learning-based techniques.

III. Pattern

A. Definition

Definition III.1 (*Pattern*). A pattern is a frequent subsequence of commit messages.

Definition III.2 (*Pattern Ratio*). Pattern ratio is the ratio of commit messages fitting with patterns to all commit messages.

Definition III.3 (*Pattern Message and Non-Pattern Message*). A pattern message is a commit message fitting with certain pattern, while a non-pattern message is a commit message which cannot fit with any pattern.

Definition III.4 (*Pattern Group and Non-Pattern Group*). Pattern group refers to the group of ***code changes*** whose ground truth commit messages fit with certain pattern, while non-pattern group refers to the group of ***code changes*** whose ground truth commit messages do not fit with any pattern.

In this work, we mainly investigate the distribution and structures of generated commit messages through the lense of pattern (i.e., frequent sequences in generated messages).

B. Pattern Collection

Our pattern collection includes two phases: *pattern mining* and *pattern merging*. *In the mining phase*, we leverage MaxSP [27] to mine raw frequent patterns from the commit messages generated by all studied learning-based techniques. MaxSP is a sequential pattern mining algorithm, which keeps the longest sequence among sequences with containment relationship. In particular, we use the implementation of MaxSP provided by the widely-used data mining library SPMF [28]. In total, we mine 28 raw patterns from commit messages generated by all studied techniques. *In the merging step*, we iteratively merge the similar patterns to obtain more general patterns. In particular, we abstract the concrete nouns used in the same position, e.g., we merge patterns "Fix typo in ...", "Fix bug in ..." and "Fix npe in ..." into a new pattern "Fix ... in ...". In addition, we merge finite alternative patterns into one pattern with "|", e.g., we merge patterns "Add ... to ..." and "Add ... for ..." to a new pattern "Add ... to|for ...". Besides, we enclose the optional contents in patterns with "[]", e.g., we merge patterns "Add ..." and "Add missing ..." to a new pattern "Add [missing] ...". For space limits, the complete merging process is in our replication package. To alleviate the threat from manual pattern merging, the first two authors independently merge patterns and an experienced colleague will be involved if they have discrepancies.

C. Our Patterns

In this way, we obtain four merged patterns and we then introduce their details as follows.

(1) Addition Pattern. The format is ***"Add [missing] ... [for|to ...]"***. The pattern is often related to the commit message where the commit adds certain element. For example, the message "*Add missing nullcheck*" fits with the Addition Pattern. In particular, Addition Pattern consists of six sub-patterns, including (1) "Add missing ... for ...", (2) "Add missing ... to ...", (3) "Add ... for ...", (4) "Add ... to ...", (5) "Add missing ...", and (6) "Add ...". These sub-patterns are ranked from complex to simple ones, thus they are mutually-exclusive. For example, a commit message fitting with the first sub-pattern, would not further be considered as fitting with the following sub-patterns. Note that, the last sub-pattern "Add ..." is actually not very general and does not broadly include many long/complex messages. In fact, most of the generated commit messages fitting with this sub-pattern follow a short format "Add" + "determiner (e.g., a/some)" + "noun" (e.g., comment/javadoc) or "Add" + "identifier" + "noun" (e.g., method/constructor), and their average length is short (i.e., 3.33 tokens).

(2) Removal Pattern. The format is ***"Remove unused|unnecessary ..."***. The pattern is often related to the commit message where the commit removes redundant elements (e.g., code, imports, and methods) in code changes. For example, "*Remove unused member*". Removal Pattern consists of 2 sub-patterns, (1) "Remove unused ...", and (2) "Remove unnecessary ...".

(3) Fix Pattern. The format is ***"Fix ... [in|when ...]"***. The pattern is often related to the commit message where the commit fixes some bugs. For example, "*Fix a buffer leak*". Fix Pattern consists of 3 sub-patterns, (1) "Fix ... in ...", (2) "Fix ... when ...", and (3) "Fix ...".

(4) Avoidance Pattern. The format is ***"Don't|do not ... [if ...]"***. The commit message fitting with this pattern states not to do some wrong behavior. Fig. 1 shows an example. Avoidance Pattern consists of 4 sub-patterns, (1) "Don't ... if ...", (2) "Do not ... if ...", (3) "Don't ...", and (4) "Do not ...".

```
@@ -137,7 +137,7 @@ public class Profiler {
          . . .
-         if (log.isDebugEnabled()) {
+         if (!mute && log.isDebugEnabled()) {
              . . .
              println(name, " took ", format(result));
```

Commit message: Don't print end message if profiler was muted

Fig. 1: Example of Avoidance Pattern

IV. Study Design

NLP metrics measure the overall performance with aggregated scores and hide the details of what generated commit messages look like. Therefore, we first conduct a comprehensive evaluation of existing techniques from not only NLP metrics but also patterns, to reflect both overall and concrete performance of generated commit messages (**RQ1**).

According to the results of RQ1, we surprisingly find that the majority of commit messages generated by existing techniques belong to some patterns, which have simple formats. Next, we explore the potential reasons for the huge increase of pattern ratio between the ground truth and generated commit messages. For a deep learning technique, dataset and model are two primary components, which may affect the final performance. In the commit message generation task, besides them, how to represent code change is another important factor. Therefore, we will explore the influence of the three factors, ***dataset***, ***input representation***, and ***model components***, on the performance of the learning-based techniques, especially on the ratio of patterns (i.e., reflected by **RQ2/3/4**).

A. Dataset

We adopt a well-established benchmark [14], [29], widely-used in existing commit message generation [9], [10], [12]–[16]. This benchmark is collected from the top 1,000 popular Java projects in GitHub, excluding the rollback and merge commits and extracting the first line from the messages [29]. The commit messages are evenly distributed in these projects and no projects concentrate the majority of the messages. Code changes not occurring in .java files and duplicated code changes are removed [14]. Finally, the dataset contains 90,661 code change and commit message pairs. Following all the existing work [12]–[16] and common practice in deep learning, we randomly split the dataset into training/validation/testing sets by an approximate ratio of 8:1:1. To reduce the influence resulting from random split, we randomly split the dataset five times with the same ratio, and compute the average results. The results of five splits are similar, e.g., the average standard deviation of Table II is only 0.30.

B. Threats to Validity

Threats to internal validity lie in the obtainment of patterns and implementation of studied techniques. To alleviate the former threat, we adopt the implementation of pattern mining algorithm provided by the widely-used [28] library SPMF. To alleviate the latter threat, we reuse the replication package of a learning-based technique if it is available and executable [15], [16], and re-implement it strictly following the paper if not. Moreover, we get similar results on these techniques as previous work [25], indicating the correctness of the re-implementation. ***Threats to external validity*** lie in the dataset, which may affect the generalization of our findings. To alleviate these threats, we adopted the widely-used benchmark. In our future, we will conduct more experiments on projects in other programming languages. ***Threats to construct validity*** lie in the metrics used to measure the performance of the techniques. To alleviate this threat, besides the widely used NLP metrics, we also propose a new perspective, i.e., pattern, to show the detailed distribution and structures of messages.

V. RQ1: Pattern Ratio

In this section, we comprehensively revisit existing techniques with both NLP metrics and patterns. In particular, this section explores three sub-questions. *(1) RQ1.a: how do the patterns of generated commit messages distribute? (2) RQ1.b: how do the pattern length and pattern ratio affect the pattern distribution? (3) RQ1.c: how do models perform for pattern and non-pattern group?* Since RQ1.a shows the pattern ratios in the generated messages are very high and they vary among different patterns, we further investigate the influential factors (e.g., length and ratio) in RQ1.b and the model performance on pattern and non-pattern messages in RQ1.c.

A. Procedure

The common process of a deep learning application is training models on the training set and testing models on the testing set. Since models have seen the right answers during training, models are supposed to achieve a similar distribution to the ground truth if being tested on the training set. To verify how well models learn the training set, we test models not only on the testing set, but also on the training set in RQ1. Following previous works [9], [10], [12]–[16], we first evaluate generated commit messages on NLP metrics. In addition, we compute the pattern ratio of commit messages generated by each model and the pattern ratio of ground truth commit messages. When judging whether one commit message belongs to a pattern, we leverage regular expression matching. Next, to compare the performance of models on generating pattern messages and non-pattern messages, we compute NLP metrics for the commit messages generated for code changes in ***pattern group*** and ***non-pattern group***. Moreover, we explore the relationship between metrics and pattern ratios. We select the generated commit messages whose BLEU is higher than a certain threshold (which is from 0 to 100 with the interval of 10) and compute their pattern ratios.

B. Results and Analysis

TABLE II: NLP metrics of various techniques.

Model		NMT	PtrGN	CODIS	CoreGen	FIRA
Training Set	BLEU	16.94	17.41	18.03	17.17	18.76
	ROUGE-L	20.41	21.17	21.81	22.54	22.83
	METEOR	13.34	14.09	14.69	16.30	16.06
Testing Set	BLEU	14.63	16.36	16.57	13.53	17.48
	ROUGE-L	17.31	19.82	19.84	17.31	21.19
	METEOR	10.91	13.01	13.08	11.72	14.58

1) RQ1.a-Pattern Ratio Distribution: Table II presents the quality of the commit messages generated by each technique for both the training and testing set. From the table, FIRA achieves the best performance on the testing set on all metrics, which is consistent with the conclusions of existing work [16]. CoreGen performs competitively on the training set, but poorly on the testing set, indicating that CoreGen has a poor generalization ability, which is neglected by previous work [15].

Table III presents the ratio of commit messages fitting with the patterns. Row "All Patterns" presents the total ratio of all patterns in the ground truth and generated commit messages, while the other rows present the ratio of single pattern. We

TABLE III: Pattern ratios of various techniques

Model		Ground Truth	NMT	PtrGN	CODIS	CoreGen	FIRA	Average Increase
Training Set	All Patterns	46.60	88.89	86.36	88.29	88.78	85.43	40.95
	Addition	15.50	19.08	23.07	22.20	21.59	21.48	5.99
	Removal	3.11	13.00	11.07	12.91	9.26	10.66	8.26
	Fix	23.88	50.87	47.35	47.38	54.84	47.03	25.62
	Avoidance	4.11	5.94	4.88	5.81	3.09	6.26	1.08
Testing Set	All Patterns	46.64	89.75	86.94	89.65	90.04	85.62	41.77
	Addition	15.36	19.05	22.81	22.66	21.15	21.48	6.07
	Removal	3.32	12.90	11.38	13.84	9.41	10.90	8.37
	Fix	23.91	52.05	48.00	47.78	56.45	47.02	26.35
	Avoidance	4.05	5.74	4.75	5.38	3.04	6.21	0.98

compute the increase of pattern ratio between the ground truth and generated commit messages, and column "Average Increase" presents the average increase of all techniques. From the first rows of the two datasets, for all techniques, most generated commit messages fit with patterns on both the training set and the testing set. For example, the total pattern ratio of CODIS on the testing set is 89.65%. The pattern ratios in the generated commit messages are much higher than the pattern ratios in the ground truth, and the average increase is 40.95% and 41.77% on the training set and the testing set respectively. We also have similar findings on sub-patterns, i.e., the ratio of sub-patterns in generated commit messages is generally higher than the ratio in the ground truth. Through the lenses of patterns, we discover the problem that NLP metrics cannot discover. Although achieving not bad performance regarding NLP metrics, the majority (˜90%) of generated commit messages fit with a few simple patterns. That is, they have the limited capacity on generating flexible commit messages. Considering the huge increase of pattern ratio between ground truth and generated commit messages, many generated commit messages fitting with patterns are wrong and they should belong to non-pattern. Here we illustrate this finding through a real case in Fig. 2. In this case, developers trim the *MVEL* scripts before compiling them. The ground truth accurately describes this behavior, which does not fit with any pattern (i.e., non-pattern message). However, all the studied techniques generate the message belonging to Fix Pattern, which is too general and contains useless information.

Finding 1: For all the learning-based techniques, the majority (i.e., ˜90%) of generated commit messages belong to simple patterns, and the ratio is much higher than that in ground truth (i.e., ˜50%).

The huge increase of pattern ratio between ground truth and generated messages is especially surprising for the ***training set***, because models have seen the "correct answers" of the code changes of the training set during training. This indicates that models cannot memorize the distribution of the training set, and non-pattern data is challenging for models to learn.

Finding 2: The huge increase of pattern ratio between generated and ground truth commit messages is especially surprising for the ***training set*** because models have seen the "correct answers" of the training set, indicating that non-pattern data is challenging for models to learn and generate correct commit messages.

```
@@ -82,7 +82,7 @@ public class MvelScriptEngineService
extends AbstractComponent implements Script
     @Override
     public Object compile(String script) {
-        return MVEL.compileExpression(script,
-            new ParserContext(parserConfiguration));
+        return MVEL.compileExpression(script.trim(),
+            new ParserContext(parserConfiguration));
     }
```

Commit message:

Ground Truth:	Trim MVEL scripts before compiling them
NMT:	Fix bug in <unk>
PtrGNCMsg:	Fix MvelScriptEngineService compile
CODISUM:	Fix bug in MvelScriptEngineService
CoreGen:	Fix the tests
FIRA:	Fix MVEL MVEL

Fig. 2: Case in which techniques generate pattern messages but the ground truth belongs to non-pattern.

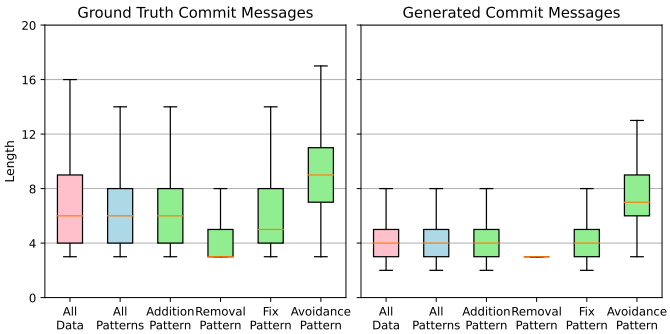

Fig. 3: Length distribution of ground truth and generated commit messages on the training set. For simplicity, we omit the outliers.

2) RQ1.b-Influence of Number and Length on Ratio of each Pattern: To explore the relationship between the ratio of a pattern in generated commit messages and the lengths of commit messages belonging to this pattern in the training set, we show the box plot of length distribution of ground truth and generated commit messages on the training set in Fig. 3. It contains two subfigures. The left and right are the ground truth and generated commit messages. Each subfigure contains several groups of data, e.g., "All Data" is the group of training data, "All Patterns" is the group of data fitting with any pattern, Addition Pattern is the group of data fitting with Addition Pattern. The vertical axis refers to the length distribution.

From the last column of Table III, the ratio of each pattern all increases between ground truth and generated commit messages, but the increase differs a lot among each specific pattern. One possible factor for different increases might be the length of commit messages of different patterns. Comparing Removal Pattern with Avoidance Pattern, they have similar pattern ratios in the training set which are the lowest among

all patterns (i.e., 3.11% and 4.11% in the first column of Table III), but Removal Pattern messages are generally shorter than Avoidance Pattern messages, according to the left part of Fig. 3. As a result, the increase of pattern ratio of Removal Pattern is higher than Avoidance Pattern (i.e., 8.37% vs 0.98% in the last column of Table III). This indicates that shorter length of certain pattern will result in higher increase of that pattern. The reason might be that shorter commit messages are easier to learn and make the model tend to generate more commit messages of that pattern. Another influence factor is the pattern ratio in the training set. Comparing Fix Pattern with Addition Pattern, they have similar length distribution, but the pattern ratio of Fix Pattern is higher in the training set (i.e., 23.88% vs 15.50%). As a result, the increase of pattern ratio of Fix Pattern is higher than Addition Pattern (i.e., 26.35% vs 6.07%). This indicates that higher ratio of certain pattern in the training set will result in higher increase of that pattern. The reason might be that higher ratio of certain pattern provides more data for models to learn and make them tend to generate messages of that pattern.

Finding 3: The increase of pattern ratio between generated and ground truth messages of different patterns differs a lot. The larger number and shorter length of messages fitting with one pattern in the training set contribute to the higher ratio of that pattern in the generated messages.

Then, comparing the two parts of Fig. 3, the generated commit messages are generally shorter than the ground truth on both all data and each pattern. For example, the decrease of first quartile, median, and third quartile of lengths of all data are 1, 2, and 4. In particular, the average length of all generated messages is less than 5. This indicates that models tend to generate short commit messages. Combining with Finding 1, the majority of generated commit messages are short and belong to simple patterns. Existing techniques have the limited capacity on generating flexible and complex commit messages.

Finding 4: Generated commit messages are shorter than the ground truth ones. Existing models mostly generate simple messages with short length and simple patterns, and have the limited capacity on generating flexible and complex ones.

3) RQ1.c- Performance for Pattern and Non-Pattern Group: Table IV presents the quality of commit messages generated for the code changes in pattern group and non-pattern group. From the table, the NLP metrics for pattern group are generally higher than non-pattern group for all techniques, and more than twice. The higher NLP metrics indicate that models perform better on pattern group than non-pattern group. In other words, the performance of existing techniques is mainly contributed by pattern group and they have a poor capacity on generating commit messages for non-pattern group.

TABLE IV: Performance for pattern and non-pattern group.

Model		BLEU	ROUGE-L	METEOR
NMT	Pattern	21.17	25.91	16.16
	Non-Pattern	8.92	9.82	6.33
PtrGN	Pattern	22.60	28.36	18.68
	Non-Pattern	10.92	12.38	8.08
CODIS	Pattern	22.79	28.36	18.90
	Non-Pattern	11.14	12.42	8.01
CoreGen	Pattern	18.88	25.21	16.95
	Non-Pattern	8.85	10.40	7.14
FIRA	Pattern	23.57	29.45	20.31
	Non-Pattern	12.17	13.99	9.59

Finding 5: The NLP metrics for pattern group are much higher than the metrics for non-pattern group, which indicates that models have a better capacity on generating pattern messages than generating non-pattern messages.

Fig. 4 presents the relationship between pattern ratio and BLEU threshold of each technique. The horizontal axis refers to the BLEU threshold, and the vertical axis refers to the pattern ratio of generated messages whose BLEU is higher than the threshold. From the figure, the curves of all techniques have the same trend which decreases first and then increases.

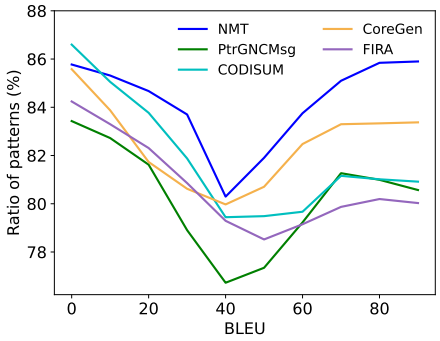

Fig. 4: The change of pattern ratio with the increase of BLEU threshold.

When the BLEU threshold is relatively low, the pattern ratio decreases with the increase of BLEU. This is because compared with non-pattern messages, more pattern messages have low BLEU and are filtered out. Many generated commit messages fitting with patterns are wrong and they should belong to non-pattern. These wrong pattern messages differ a lot from the ground truth and have rather low BLEU. If the BLEU threshold increases, wrong pattern messages with low-quality will be filtered out. When the threshold increases from 0 to where pattern ratio is lowest, in the commit messages which are filtered out, on average, 45% are predicted as pattern messages but they should belong to non-pattern, and only 3% are predicted as non-pattern messages but they should belong to certain pattern. More pattern messages are wrongly

predicted and filtered out than non-pattern messages, so the pattern ratio decreases.

When the BLEU threshold is relatively high, the pattern ratio increases with the increase of BLEU. When increasing the BLEU threshold continuously, the quality of remaining messages is increasingly higher. According to Finding 5, models perform better on generating commit messages fitting with pattern than non-pattern. Therefore, with the further increase of BLEU, the ratio of pattern messages will increase.

> **Finding 6:** When increasing the BLEU threshold, the pattern ratio decreases first and then increases. The reason for the decrease is that the wrongly-generated pattern messages with low quality are filtered out, while the reason for the increase is that models perform better on generating pattern messages.

VI. RQ2: Dataset

In this section, we explore the influence of the pattern ratio in datasets by two sub-questions. *(1) RQ2.a: how does the pattern ratio in the training set influence the pattern ratio in the generated commit messages? (2) RQ2.b: how does the pattern ratio in the training set influence the model performance on pattern group and non-pattern group?* These two sub-questions investigate the impact of the pattern ratio in the training set by two aspects.

A. Procedure

To explore the relationship between pattern ratio in generated commit messages and the training set, we reduce the pattern ratio in the training set and retrain the model with the modified training set. The reduction procedure is as follows. We divide the training set into $n+1$ groups, including n groups whose commit messages belong to n pre-defined patterns in Section III and one group containing non-pattern data. We reduce the number of data belonging to n patterns and increase the number of non-pattern data to keep the total number of data unchanged. In particular, we randomly reduce the number of each pattern by 10% to 90% and the interval is 10%. In addition, we randomly select non-pattern data with the number of removed pattern data to keep the total number unchanged. We retrain models using the modified training set and test the retrained models using the original testing set.

B. Results and Analysis

1) RQ2.a-Influence of Pattern Ratio in the Training Set: Fig. 5 presents the change of pattern ratio in the generated commit messages with the decrease of pattern ratio in the training set. The horizontal axis refers to the decrease ratio of pattern messages in the training set and the vertical axis refers to the pattern ratio in the generated commit messages. From this figure, with the decrease of patterns in the training set, the pattern ratio in the generated commit messages decreases continuously. This indicates that the pattern ratio in the generated commit messages is positively associated with the pattern ratio in the training set, and models are inclined to generating commit messages which appear frequently in the training set.

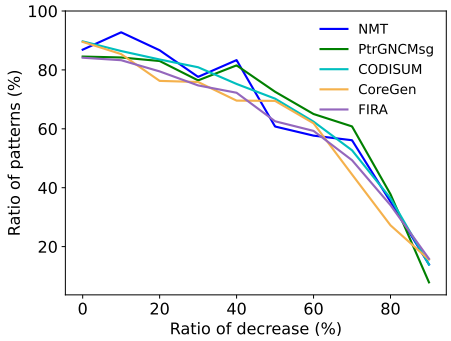

Fig. 5: The change of pattern ratio in the generated commit messages with the decrease of pattern ratio in the training set.

> **Finding 7:** With the decrease of pattern ratio in the training set, the pattern ratio in generated commit messages decreases continuously. The pattern ratio in generated commit messages is positively associated with the ratio in the training set.

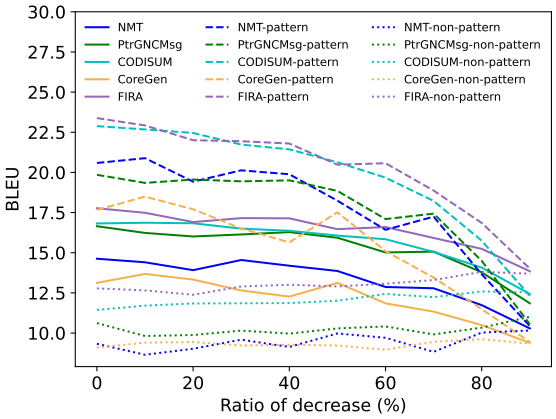

Fig. 6: The change of BLEU in the generated commit messages with the decrease of pattern ratio in the training set.

2) RQ2.b-Change of Performance on Different Groups: Fig. 6 presents the change of BLEU for different groups with the decrease of pattern ratio in the training set. Besides the total commit messages, we also show the results of commit messages generated for code changes of ***pattern group*** and ***non-pattern group*** to see the influence of pattern ratio on different groups. From this figure, with the decrease of pattern ratio in the training set, the BLEU of total generated commit messages decreases continuously. In addition, the BLEU for pattern group decreases and BLEU for non-pattern group remains almost unchanged. Therefore, the decrease of BLEU for total commit messages results from the decrease of BLEU

for pattern group. The reason is that, with the decrease of pattern ratio in the training set, pattern data become gradually insufficient for models to learn. The capacity of models on generating pattern messages decreases and the quality of generated commit messages decreases.

On the other hand, although we increase the ratio of non-pattern data in the training set and correspondingly the ratio of non-pattern messages in the generated messages increases, the quality of non-pattern messages does not improve and their BLEU is always lower than BLEU of pattern messages. In other words, the high ratio of pattern messages is not the reason for the poor quality of non-pattern messages and non-pattern data are inherently challenging for models to learn.

Finding 8: With the decrease of pattern ratio in the training set, the BLEU of generated commit messages decreases. The decrease results from the BLEU decrease of pattern group, while the BLEU of non-pattern group remains almost unchanged, which indicates that non-pattern data are inherently challenging for models to learn.

VII. RQ3: Input Representation

In this section, we explore the influence of input code change representation by two sub-questions. *(1) RQ3.a: how do the models perform with only change marks? (2) RQ3.b: how do the models perform for pattern and non-pattern groups with only change marks?* These two sub-questions investigate the overall performance and separate performance when input code changes are represented with only change marks.

A. Procedure

In code changes, each line starts with a change mark, i.e., "+", "-" and " ", indicating the line is added, deleted or unchanged. Code changes consist of two types of tokens, code text tokens and change mark tokens. For example, given a code change, "`- a = 0; + a = 1;`", code text tokens are "`a`", "`=`", "`0`", "`;`", "`a`", "`=`", "`1`", "`;`" and mark tokens are "`-`", "`+`". We propose a simplified representation which replaces each token in code changes with the corresponding mark, called ***mark representation***. Mark representation contains only change information of code changes, not concrete code text information. In the previous example, the mark representation is "`- - - - + + + +`". To explore the impact of input representation, we train a model with mark&code representation (default), and only mark representation respectively, and compare their performance.

B. Results and Analysis

1) RQ3.a-Competitive Performance with Only Mark Representation: Table V presents the results of each technique trained by the dataset with different input representation. Row "Code&Mark" presents the results when using code&mark representation (i.e., the default setting), while Row "Only Mark" presents the results using only mark representation.

From the table, the performance of using only mark representation is close to using code&mark representation. For example, the BLEU of FIRA using mark and code&mark representation are 15.42 and 17.48, respectively. FIRA with only mark representation even outperforms NMT and CoreGen with code&mark representation. We further remove marks from code&mark representation and reserve only code text in implementing a variant of each technique, and find that this variant achieves poorer performance than the corresponding technique using only mark representation. That is, using only mark representation can achieve competitive performance and the code text of code changes brings limited gain to the performance. This is surprising because mark representation is a very summarized representation including only three symbols (i.e., "+", "-", and " "), and has no syntax or semantics.

TABLE V: Results of models when using various input representation

Model		BLEU	ROUGE-L	METEOR
NMT	Code&Mark	14.63	17.31	10.91
	Only Mark	12.80	15.49	8.62
PtrGN	Code&Mark	16.36	19.82	13.01
	Only Mark	13.17	15.45	9.88
CODIS	Code&Mark	16.57	19.84	13.08
	Only Mark	14.24	16.46	10.46
CoreGen	Code&Mark	13.53	17.31	11.72
	Only Mark	10.28	13.35	8.21
FIRA	Code&Mark	17.48	21.19	14.58
	Only Mark	15.42	18.07	11.62

TABLE VI: Ratio of each pattern with different mark types

Mark Type		Addition Pattern	Removal Pattern	Fix Pattern	Avoidance Pattern	Non Pattern
"+"&" "	Ground Truth	**39.14**	0.06	13.78	3.05	**43.98**
	Code&Mark	**60.24**	0	27.12	4.35	**8.29**
	Only Mark	80.91	0	16.47	1.58	1.04
"-"&" "	Ground Truth	0.53	**24.01**	10.21	2.79	**62.47**
	Code&Mark	0	**68.70**	1.67	1.67	**27.96**
	Only Mark	0	88.62	0.21	0.05	11.11
"+"&"-"&" "	Ground Truth	9.51	1.53	**29.93**	4.45	**54.58**
	Code&Mark	11.60	8.34	**61.37**	6.24	**12.45**
	Only Mark	9.17	11.44	74.75	0.33	4.31

Next, we present pattern ratios with different mark types to further explore the impact of marks. We divide the code changes into three groups according to the mark type, including (1) "+"&" ", (2) "-"&" ", and (3) "+"&"-"&" ". For the code changes of each mark type, we present the ratio of each pattern in corresponding commit messages in Table VI. The row "Ground Truth" presents the pattern ratios in the ground truth; the rows "Code&Mark" and "Only Mark" present the pattern ratios when using code&mark representation and only mark representation, respectively. Each column represents the results in different patterns and non-pattern.

From this table, for the code changes of each mark type, in the corresponding ground truth commit messages, non-pattern occupies the highest ratio (˜50%). Besides non-pattern, for each mark type, there exists one pattern with the highest ratio, and we call it *mark-related pattern*. The mark-related pattern

has strong correlation with the mark type, e.g., for the mark type with "+"&" " marks, Addition Pattern is the mark-related pattern (ratio is 39.14%). This observation is expected because code changes with only "+" and " " marks usually result from developers' intention on adding something. Similarly, the mark-related patterns of mark type "-"&" " and "+"&"-"&" " are Removal Pattern and Fix Pattern, respectively. As the mark-related pattern has strong correlation with marks, with only marks, models tend to generate correct commit messages belonging to that pattern. However, for each mark type, non-pattern occupies the highest ratio, so many non-pattern data share similar mark sequences to the mark-related pattern data, which indicates that marks alone may not be sufficient for correct commit message generation for non-pattern data. In other words, besides code change marks, models need to capture the semantics and syntax of code text to generate commit messages of non-pattern. On the other hand, models generate much higher ratio of commit messages fitting with the mark-related pattern, and much lower ratio of non-pattern, indicating many generated commit messages of mark-related patterns are wrong and they should belong to non-pattern. To sum up, we hypothesize that the existing techniques mainly leverage marks in message generation, instead of capturing the semantics and syntax of code text, and thus most generated commit messages belong to simple patterns.

For each mark type, the quality of commit messages generated for code changes whose ground truths belong to the mark-related pattern is much better than non-pattern. For example, for the mark type "+"&" ", BLEU for Addition Pattern and non-pattern are 25.82 and 7.90, respectively. Moreover, in the cases whose BLEU exceeds 50, Addition Pattern and non-pattern occupy 86.06% and 8.53% respectively, indicating that models can mainly deal with the cases where commit messages have strong correlation with change marks. When using only mark representation, these cases can also be solved. With only mark representation, BLEU on Addition Pattern is 23.25, similar to code&mark. Therefore, only mark representation can achieve similar performance to code&mark representation.

Finding 9: Models fail to capture the syntax and semantics of input code and mainly leverage marks to generate commit messages, so they generate high ratio of pattern messages and have competitive performance even when the inputs are only represented by change marks.

2) RQ3.b-Performance for Pattern and Non-Pattern Group: Table VII presents the NLP metrics of the commit messages generated for the code changes in ***pattern group*** and ***non-pattern group***, when training models with various input representation. Rows "Code&Mark" and "Only Mark" show the results using code&mark and mark representation. The left part of each column is NLP metric values and the right (Decl) is the decrease ratio from code&mark representation. In this table, the decrease ratio from code&mark representation to mark representation for ***non-pattern group*** is higher than ***pattern group***. For example, the decrease ratio of BLEU of CoreGen for non-pattern group and pattern group are 32% and 20% respectively. That is, without code text, models lose more performance on non-pattern group than pattern group. Mark representation contains limited information, which lacks syntax and semantics. However, non-pattern commit messages are of flexible formats, and more difficult to generate, so when using only mark representation, models have a poorer capacity on generating non-pattern commit messages.

TABLE VII: Results for pattern group and non-pattern group when using various input representation

Model			BLEU/Decl		ROUGE-L/Decl		METEOR/Decl	
NMT	Code&Mark	Pattern	21.17	-	25.91	-	16.16	-
		Non-Pattern	8.92	-	9.82	-	6.33	-
	Only Mark	Pattern	18.92	11%	24.05	7%	13.51	16%
		Non-Pattern	7.37	17%	7.88	20%	4.27	33%
PtrGN	Code&Mark	Pattern	22.60	-	28.36	-	18.68	-
		Non-Pattern	10.92	-	12.38	-	8.08	-
	Only Mark	Pattern	18.91	16%	23.32	18%	15.14	19%
		Non-Pattern	8.07	26%	8.47	32%	5.21	36%
CODIS	Code&Mark	Pattern	22.79	-	28.36	-	18.90	-
		Non-Pattern	11.14	-	12.42	-	8.01	-
	Only Mark	Pattern	20.06	12%	24.32	14%	15.90	16%
		Non-Pattern	9.06	19%	9.48	24%	5.63	30%
CoreGen	Code&Mark	Pattern	18.88	-	25.21	-	16.95	-
		Non-Pattern	8.85	-	10.40	-	7.14	-
	Only Mark	Pattern	15.02	20%	21.25	16%	13.26	22%
		Non-Pattern	6.06	32%	6.32	39%	3.72	48%
FIRA	Code&Mark	Pattern	23.57	-	29.45	-	20.31	-
		Non-Pattern	12.17	-	13.99	-	9.59	-
	Only Mark	Pattern	21.21	10%	25.82	12%	16.82	17%
		Non-Pattern	10.27	16%	11.18	20%	7.00	27%

Finding 10: Using only mark representation loses more performance on non-pattern group than pattern group. The limited information contained in mark is not sufficient for generating flexible and complex non-pattern messages.

VIII. RQ4: Model Component

In this section, we explore the influence of model components by three sub-questions. (1) RQ4.a: how does the attention mechanism affect the effectiveness? (2) RQ4.b: how does the copy mechanism affect the effectiveness? (3) RQ4.c: how does the anonymization mechanism affect the effectiveness?

A. Procedure

As shown in Table I, the components of the studied models include attention, mechanism, and anonymization mechanism. To explore the impact of each component on the pattern ratios and NLP metrics, we compare the performance of the default model and its variants with different components removed. Note that CoreGen leverages the transformer [30] (whose attention cannot be removed) and has no other components, so we exclude CoreGen in this RQ.

B. Results and Analysis

Table VIII presents the quality of commit messages generated by each approach and their variants removing different components. Row "Orignal" presents the measurement results of commit messages generated by the model while the other

TABLE VIII: Results of models removing each component

Model		BLEU	ROUGE-L	METEOR	Pattern Ratio
NMT	Original	14.63	17.31	10.91	89.75
	Attention mechanism	13.66	16.24	10.09	91.37
PtrGNCMsg	Original	16.36	19.82	13.01	86.94
	Attention mechanism	15.81	18.81	12.13	86.95
	Copy mechanism	15.97	18.97	12.31	88.37
CODISUM	Original	16.57	19.84	13.08	89.65
	Attention mechanism	15.93	18.88	12.30	93.50
	Copy mechanism	16.29	19.42	12.79	89.35
	Anonymization	16.03	19.38	12.20	88.92
FIRA	Original	17.48	21.19	14.58	85.62
	Copy mechanism	17.13	20.73	14.13	83.58
	Anonymization	17.19	21.31	14.40	84.90

rows represents the results of the model without that component. Column "Pattern Ratio" presents the ratio of commit messages fitting with the patterns to all generated messages.

1) RQ4.a-Attention Mechanism: From the table, the attention component influences the quality of generated commit messages. When the attention component is removed, all the NLP metric values decrease. In addition, the ratio of pattern-matching commit messages in the generated commit messages increases. One potential reason might be that without the attention mechanism, the capacity of models decreases and the models generate more commit messages with simple patterns.

TABLE IX: Attention weights on marks

Model	Ratio of Number		Attention Weights	
	Changed	Unchanged	Changed	Unchanged
NMT	38.12	61.88	46.37	53.63
PtrGN			64.51	35.49
CODIS			56.90	43.10
FIRA			66.42	33.58

To further explore the attention mechanism, we compute attention distribution, and show the results in Table IX. We get attention on each input token, which indicates to what extent the decoder concentrates on this token. Columns "Ratio of Number" refer to the ratio of the number of input tokens with changed/unchanged marks to the number of total tokens, while Columns "Attention Weights" refer to the attention weights of models on input tokens with changed/unchanged marks.

From this table, the ratio of input tokens with changed marks is lower than the ratio of tokens with unchanged mark (i.e., 38.12% vs 61.88%). Although the ratio of input tokens with changed marks is lower, the attention weight on tokens with changed marks is higher than that on tokens with unchanged marks. This indicates that the models focus on changed tokens more, and changed tokens are more important for capturing the intention behind code changes.

Finding 11: When generating commit messages, the decoder pays more attention on changed tokens (i.e.,"+" and "-") than unchanged tokens, indicating that changed tokens are more important for capturing the intention behind code changes.

Next, we explore which input tokens a learning-based technique concentrates the most by identifying its top-10 concentrated tokens as follows. Whenever a learning-based technique generates an output token, we compute the attention on input tokens during this process and select the top-10 input tokens with the highest attention weights, which are regarded as concentrated tokens; then we calculate the occurring times of all concentrated tokens through the whole commit generation process of a technique and select the top-10 concentrated tokens with the largest occurring times.

TABLE X: Top-10 input tokens models pay most attention to

Model	Top-10 Input Tokens
NMT	$\langle nl \rangle$, $\langle eos \rangle$, public, class, $\langle nb \rangle$, $\langle add \rangle$, $\langle delete \rangle$, $\langle start \rangle$, ., {
PtrGN	$\langle extend-1 \rangle$, class, $\langle extend-3 \rangle$ $\langle extend-4 \rangle$, $\langle extend-2 \rangle$, $\langle extend-5 \rangle$, $\langle eos \rangle$, $\langle extend-6 \rangle$, $\langle extend-7 \rangle$, return
CODIS	c0, n0, n2, n3, n1, n4, f0, n5, a0, $\langle eos \rangle$
FIRA	n0, n2, n1, import, n3, get, $\langle nl \rangle$, c0, if, f0

Table X presents the top-10 concentrated tokens the models pay most attention to. From the table, NMT pays attention to special tokens (e.g., $\langle nl \rangle$ and $\langle add \rangle$) and keywords (e.g., public and class), which are anchors of code changes and contain significant information. PtrGN pays attention to the extended vocabulary words (i.e., $\langle extend{-}x \rangle$), which are used to represent the out-of-vocabulary identifiers and $\langle extend{-}x \rangle$ represents a different OOV word in different code changes. Identifiers are key components of code changes and appear in commit messages frequently, so extended words are paid much attention to. Similarly, CODIS and FIRA leverage anonymization to represent identifiers as placeholders, so they pay more attention to placeholders (e.g., c0 and n0); NMT does not leverage anonymization and identifiers do not have uniform forms, and thus identifiers are not within the top-10 input tokens. In addition, FIRA pays attention to the keywords (e.g., import and if).

2) RQ4.b-Copy Mechanism: The copy mechanism is effective for commit message generation. When a copy mechanism is removed from the model, all NLP metric values decrease. The metric values for both pattern group and non-pattern group decrease. In addition, when removing the copy mechanism, the pattern ratio decreases for most techniques.

3) RQ4.c-Anonymization Mechanism: After removing anonymization, due to the limitation of vocabulary size, many identifiers occurring less than the threshold are excluded from the vocabulary and replaced by $\langle unk \rangle$. Therefore, the model generates more $\langle unk \rangle$ in the commit messages and cannot generate identifiers correctly. From the table, after removing anonymization, all NLP metric values decrease, indicating that anonymization is beneficial for commit message generation. Without anonymization, the pattern ratio decreases.

Finding 12: Model components have different influence on the ratio of pattern-matching commit messages, in particular, without attention mechanism, the ratio increases.

IX. Discussion

In this section, we discuss our findings and their implications.

The majority of automatically generated commit messages belong to simple patterns (Finding 1-2). Finding 1 and Finding 2 show that both in the testing and training sets, the majority (i.e., ˜90%) of generated commit messages belong to simple patterns and such a ratio is much higher than that in the ground truth (i.e., ˜50%).

Based on the findings, we have the following implications. (1) *Pattern-based metrics are necessary for evaluating commit message generation techniques.* Our findings indicate a huge gap between evaluating existing techniques by NLP metrics and by patterns. Pattern-based metrics can capture the structural details of generated messages, which are missed by existing NLP metrics; while NLP metrics show the semantic similarity between generated messages and the ground truth. These two metrics are complementary; (2) *A two-stage generation paradigm might be a potential direction.* Since our findings indicate that models generate more messages of certain patterns, a potential direction is to generate commit messages in two stages: first generating an abstract and pattern-based representation, and then concretizing the intermediate patterns.

Existing techniques tend to generate commit messages with short length (Finding 3-4). Finding 3 and Finding 4 show that the generated commit messages are shorter than the ground truth ones and the pattern with shorter length is more inclined to be generated by the existing techniques.

These findings further imply that the existing models tend to generate simple messages with short length. One potential direction to address this limitation is to include some constraints during the future commit message generation, such as adding extra loss to the commit messages of short length and simple patterns during training, which could encourage the models to generate more flexible and longer commit messages.

Generating non-pattern commit messages should be an important future direction (Finding 5-8). Our findings (i.e., Finding 5 and Finding 6) further show that the performance of existing models is much better in generating pattern messages than non-pattern messages. In addition, Finding 7 and Finding 8 further show the non-pattern messages are inherently more challenging for models to learn, since increasing the ratio of non-patterns in the training set fails to improve the quality of generated non-pattern commit messages in the testing set.

These findings actually indicate that the overall "decent" performance of existing models is mainly contributed by their good performance in generating those pattern messages. In other words, generating non-pattern messages is currently a challenging bottleneck in existing techniques and is definitely an important future direction to explore.

Syntax and semantics in code changes should be better captured (Finding 9-12). Finding 9 and Finding 10 show that models have comparable performance even with input code changes only represented as change marks, while such an observation is more evident on pattern messages.

These findings actually imply that existing models fail to fully capture the syntax and semantics in code changes, especially when generating non-pattern messages. There are two potential directions to future better capture syntax and semantics in code changes: (1) *Better code change representation.* Finding 11 shows that changed tokens are more important for commit message generation while too many unchanged tokens are a disturbance for models to capture the syntax and semantics. Thus, the code change representation that reserves all changed tokens but a subset of important unchanged tokens (e.g., class names and method names), might help models to pay more attention to changed tokens; (2) *Specific model design for code changes*. Finding 12 indicates the components have different influence on the pattern ratio and are all relevant to the high pattern ratio. Given the current models are mainly adopted from the NLP domain, a more specific model design for code changes might help better capture the syntax and semantics, such as conducting the attention mechanism within changed tokens and unchanged tokens respectively and then merging them with different weights.

Implications for developers. In addition to the implications mentioned above for researchers, we briefly discuss our implications for developers. First, when developers are using existing automated commit message generation tools, our findings could suggest different user expectations on generating different commit messages. In particular, if developers need some commit messages of simple patterns, it is more feasible for them to trust and directly use these automated messages; but if they need something more flexible and more complicated, the automated messages might not be satisfactory. Second, when developers write commit messages during their development activities, our findings encourage them to present complex messages in a more structured and uniform format. This is because our findings show that commit messages of patterns are easier for models to learn and commit messages of non-pattern are more challenging, and these well-structured messages would also serve as high-quality training data for future learning-based techniques.

X. Conclusion

In this work, we conduct a comprehensive study on existing learning-based commit message generation techniques from a novel perspective pattern. We find that most generated commit messages (˜90%) belong to simple patterns, much more than the ground truth (˜50%). Inspired by this observation, we explore the reason for so many pattern messages by varying the dataset, input representation, and components of models.

XI. Data Availability

Our package is available at [31], which contains the dataset, commit messages, and scripts for reproduction.

REFERENCES

[1] L. Moreno, G. Bavota, M. Di Penta, R. Oliveto, A. Marcus, and G. Canfora, "Arena: an approach for the automated generation of release notes," *IEEE Transactions on Software Engineering*, vol. 43, no. 2, pp. 106–127, 2016.

[2] S. S. Nath and B. Roy, "Towards automatically generating release notes using extractive summarization technique," in *International Conference on Software Engineering & Knowledge Engineering, SEKE*, 2021, pp. 241–248.

[3] J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" *ACM sigsoft software engineering notes*, vol. 30, no. 4, pp. 1–5, 2005.

[4] S. Kim, T. Zimmermann, K. Pan, E. James Jr *et al.*, "Automatic identification of bug-introducing changes," in *21st IEEE/ACM international conference on automated software engineering (ASE'06)*. IEEE, 2006, pp. 81–90.

[5] R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen, "Boa: A language and infrastructure for analyzing ultra-large-scale software repositories," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 422–431.

[6] R. P. Buse and W. R. Weimer, "Automatically documenting program changes," in *Proceedings of the IEEE/ACM international conference on Automated software engineering*, 2010, pp. 33–42.

[7] L. F. Cortés-Coy, M. Linares-Vásquez, J. Aponte, and D. Poshyvanyk, "On automatically generating commit messages via summarization of source code changes," in *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*. IEEE, 2014, pp. 275–284.

[8] J. Shen, X. Sun, B. Li, H. Yang, and J. Hu, "On automatic summarization of what and why information in source code changes," in *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1. IEEE, 2016, pp. 103–112.

[9] T. Hoang, H. J. Kang, D. Lo, and J. Lawall, "Cc2vec: Distributed representations of code changes," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 518–529.

[10] Z. Liu, X. Xia, A. E. Hassan, D. Lo, Z. Xing, and X. Wang, "Neural-machine-translation-based commit message generation: how far are we?" in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 373–384.

[11] Y. Huang, N. Jia, H.-J. Zhou, X.-P. Chen, Z.-B. Zheng, and M.-D. Tang, "Learning human-written commit messages to document code changes," *Journal of Computer Science and Technology*, vol. 35, no. 6, pp. 1258–1277, 2020.

[12] S. Jiang, A. Armaly, and C. McMillan, "Automatically generating commit messages from diffs using neural machine translation," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2017, pp. 135–146.

[13] Q. Liu, Z. Liu, H. Zhu, H. Fan, B. Du, and Y. Qian, "Generating commit messages from diffs using pointer-generator network," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 299–309.

[14] S. Xu, Y. Yao, F. Xu, T. Gu, H. Tong, and J. Lu, "Commit message generation for source code changes," in *IJCAI*, 2019.

[15] L. Y. Nie, C. Gao, Z. Zhong, W. Lam, Y. Liu, and Z. Xu, "Coregen: Contextualized code representation learning for commit message generation," *Neurocomputing*, vol. 459, pp. 97–107, 2021.

[16] J. Dong, Y. Lou, Q. Zhu, Z. Sun, Z. Li, W. Zhang, and D. Hao, "FIRA: fine-grained graph-based code change representation for automated commit message generation," in *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*. ACM, 2022, pp. 970–981.

[17] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[18] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, 2015, pp. 1412–1421.

[19] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, 2015, pp. 2692–2700.

[20] A. See, P. J. Liu, and C. D. Manning, "Get to the point: Summarization with pointer-generator networks," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, 2017, pp. 1073–1083.

[21] J. Li, Y. Wang, M. R. Lyu, and I. King, "Code completion with neural attention and pointer networks," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, 2018, pp. 4159–4165.

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

[23] C.-Y. Lin and F. J. Och, "Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics," in *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, 2004, pp. 605–612.

[24] S. Banerjee and A. Lavie, "Meteor: An automatic metric for mt evaluation with improved correlation with human judgments," in *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, 2005, pp. 65–72.

[25] W. Tao, Y. Wang, E. Shi, L. Du, S. Han, H. Zhang, D. Zhang, and W. Zhang, "On the evaluation of commit message generation models: An experimental study," in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2021, pp. 126–136.

[26] Y. Tian, Y. Zhang, K. Stol, L. Jiang, and H. Liu, "What makes a good commit message?" in *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*. ACM, 2022, pp. 2389–2401. [Online]. Available: https://doi.org/10.1145/3510003.3510205

[27] P. Fournier-Viger, C.-W. Wu, and V. S. Tseng, "Mining maximal sequential patterns without candidate maintenance," in *International Conference on Advanced Data Mining and Applications*. Springer, 2013, pp. 169–180.

[28] P. Fournier-Viger, J. C. Lin, A. Gomariz, T. Gueniche, A. Soltani, Z. Deng, and H. T. Lam, "The SPMF open-source data mining library version 2," in *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2016, Riva del Garda, Italy, September 19-23, 2016, Proceedings, Part III*, ser. Lecture Notes in Computer Science, B. Berendt, B. Bringmann, É. Fromont, G. C. Garriga, P. Miettinen, N. Tatti, and V. Tresp, Eds., vol. 9853. Springer, 2016, pp. 36–40. [Online]. Available: https://doi.org/10.1007/978-3-319-46131-1_8

[29] S. Jiang and C. McMillan, "Towards automatic generation of short summaries of commits," in *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. IEEE, 2017, pp. 320–323.

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[31] "Replication package," https://doi.org/10.5281/zenodo.7042270, 2022.