# RNSum: A Large-Scale Dataset for Automatic Release Note Generation via Commit Logs Summarization

**Hisashi Kamezawa**[1] **Noriki Nishida**[2] **Nobuyuki Shimizu**[3]
**Takashi Miyazaki**[3] **Hideki Nakayama**[1]
[1] The University of Tokyo
[2] RIKEN Center for Advanced Intelligence Project
[3] Yahoo Japan Corporation
hisas@nlab.ci.i.u-tokyo.ac.jp
noriki.nishida@riken.jp
{nobushim, takmiyaz}@yahoo-corp.jp
nakayama@ci.i.u-tokyo.ac.jp

## Abstract

A release note is a technical document that describes the latest changes to a software product and is crucial in open source software development. However, it still remains challenging to generate release notes automatically. In this paper, we present a new dataset called RNSum, which contains approximately 82,000 English release notes and the associated commit messages derived from the online repositories in GitHub. Then, we propose classwise extractive-then-abstractive/abstractive summarization approaches to this task, which can employ a modern transformer-based seq2seq network like BART and can be applied to various repositories without specific constraints. The experimental results on the RNSum dataset show that the proposed methods can generate less noisy release notes at higher coverage than the baselines. We also observe that there is a significant gap in the coverage of essential information when compared to human references. Our dataset and the code are publicly available.

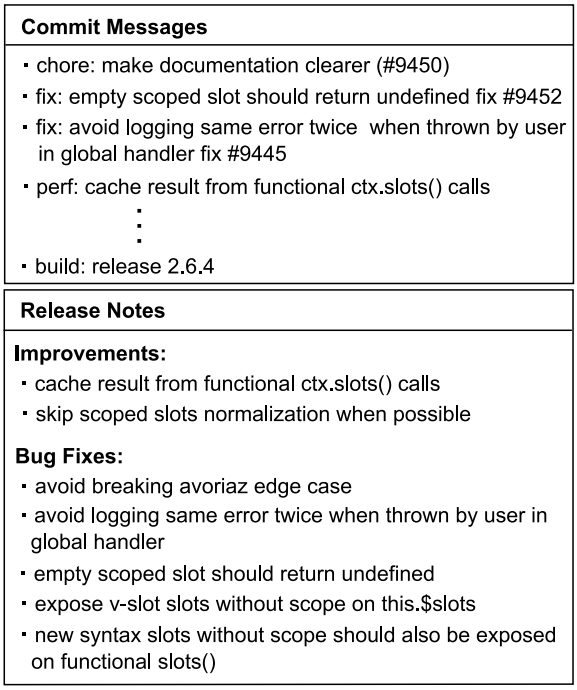


Figure 1: An example data in RNSum; this example is derived from the release of tag v2.6.4 in https://github.com/vuejs/vue.

## 1 Introduction

Recently, there has been considerable interest in applying natural language processing (NLP) techniques to support software development (Iyer et al., 2016; Yin and Neubig, 2018; Panthaplackel et al., 2020). One such task involves the automatic generation of *release notes*. A release note is a technical document that describes the latest changes to a software product, which is necessary for software developers to adjust their codes accurately for using the updated software. Since release notes are time-consuming to write manually, several studies have been done to explore automatic release note generation. Moreno et al. (2014) proposed ARENA, an automatic release notes generator, which first extracts and summarizes the changes in the source code and then integrates them with the information provided by version-control and issue-tracking systems. Pokorný (2020) developed Glyph, which classifies commit messages into predefined release-note categories (e.g., Features, Bug Fixes) using pre-trained word embeddings and produces the categorized commit messages as the final release note.

Despite the progress reported in these previous studies, usable release note generators are far from realization. We attribute this difficulty mainly to



*Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*
*Volume 1: Long Papers*, pages 8718 - 8735

two problems. First, the existing resources for automatic release note generation are scarce; for example, Glyph was trained on only 5,000 commit messages, which is too little for obtaining a sufficiently generalized model. Second, the existing methods have limited applicability; for example, ARENA requires an issue tracker hosted on Jira, thus preventing it from being used for most GitHub repositories. Also, Glyph's predefined release-note categories do not include deprecations and removals, which are often indispensable in release notes.

To alleviate the above problems, we introduce RNSum, a new large-scale dataset for automatic release note generation via commit logs summarization. An example data in RNSum is shown in Figure 1. RNSum consists of approximately 82,000 release notes derived from online repositories on GitHub. The contents of each release note are further categorized into four release-note classes: (1) Features, (2) Improvements, (3) Bug Fixes, and (4) Deprecations+ (Deprecations, Removals, and Breaking Changes). The release notes are associated with the commit messages that were used by human maintainers to write the release notes.

The difficulty of this task is that there is no explicit alignment between each commit message and the release note categories. For example, in Figure 1, the first commit message ("chore: make documentation clearer (#9450)") is not reflected in the release notes. In contrast, the second commit message ("fix: empty scoped slot should return undefined fix #9452") is reflected as the third release note in the Bug Fixes class. We propose two approaches to this task: Classwise Extractive-then-Abstractive Summarization (CEAS) and Classwise Abstractive Summarization (CAS) models, which learn to produce categorized release notes given unlabeled commit messages in extractive and abstractive manners, respectively. The two proposed models can leverage modern transformer-based sequence-to-sequence (seq2seq) architectures (e.g., BART (Lewis et al., 2020)) and can be used for various repositories without any special constraints.

We evaluate the proposed models and the previous systems on the RNSum dataset and report that our approaches generate less noisy release notes at higher coverage than the baselines given only unlabeled commit messages. We also perform human evaluations carefully to assess how well the systems could generate release notes in terms of quality (precision) and coverage (recall), revealing that there still remains a significant gap in the coverage when compared to human references. Our dataset and the source code are publicly available.[1]

| | CM | | RN | | |
|---|---|---|---|---|---|
| Dataset | Text | Class | Text | Class | Size |
| Mauczka et al. (2015) | ✓ | ✓ | | | 967 |
| Levin and Yehudai (2017) | ✓ | ✓ | | | 1,151 |
| Safdari (2018) | ✓ | ✓ | | | 3,377 |
| RNSum (ours) | ✓ | | ✓ | ✓ | 81,996 |

Table 1: Comparison of RNSum with the existing datasets on commit logs. CM and RN denote the commit message and release note, respectively.

## 2 Task Formulation

Here, we define the automatic release note generation task. The input is a set of commit messages (sentences), $\boldsymbol{x} = \{x_1, \ldots, x_n\}$. Given the input commit messages $\boldsymbol{x}$, our goal is to generate labeled release notes $\boldsymbol{y}_c$ for each predefined release-note class $c \in \mathcal{C}$. Each labeled release note is a collection of sentences, i.e., $\boldsymbol{y}_c = y_{c,1}, y_{c,2}, \ldots$. According to Moreno et al. (2014), the major contents of most release notes can be categorized into the following classes: Fixed Bugs, New Features, New Code Components (CC), Modified CC, etc. Based on their observations, we define the release-note classes $\mathcal{C}$ comprising the following four categories: Features (F), Improvements (I), Bug Fixes (B), and Deprecations+ (D; Deprecations, Removals, and Breaking Changes).[2] Our classes do not include Refactoring, Document Changes, and Library Updates because most of the maintainers on GitHub omitted these changes in their release notes.

Our dataset can be interpreted as a collection of quintuples, namely $\{\boldsymbol{x}, \boldsymbol{y}_\mathrm{F}, \boldsymbol{y}_\mathrm{I}, \boldsymbol{y}_\mathrm{B}, \boldsymbol{y}_\mathrm{D}\}$. For simplicity, we also use $(\boldsymbol{x}, \boldsymbol{y})$ instead of the quintuple representation. It is worth noting that the labeled release notes $\boldsymbol{y}_c$ can be empty. For example, in Figure 1, the software update is related to improvements and bug fixes. Thus, the release note contains only the Improvements and Bug Fixes classes, and $\boldsymbol{y}_\mathrm{F} = \boldsymbol{y}_\mathrm{D} = \emptyset$.

## 3 Related Work

**Release Note Generation** Automatic release note generation has been studied by several research groups. Moreno et al. (2014) proposed ARENA, which transforms the extracted source-code changes into the corresponding natural language release notes. However, ARENA relies on a versioning system and an issue tracker hosted on Jira, which makes it difficult or even impossible to use in a variety of software projects, especially those hosted on GitHub. Klepper et al. (2016) proposed a semi-automatic algorithm to generate the release notes depending on expected types of readers, e.g., team members, customers, and users. However, no experiments were reported in their work. Recently, a publicly available release note generator, Glyph (Pokorný, 2020), was developed. Glyph is a simple learning-based model that classifies each input commit message into one of five labels: Features, Bug Fixes, Improvements, Non-functional, and Other. These categorized commit messages are then used as the final release notes. The Glyph model was trained on 5,000 commit logs using Facebook's fastText framework (Joulin et al., 2017). We summarize the comparison of our dataset with the three data sources used in Glyph in Table 1. These existing datasets annotate only the commit message with the release note classes, making it difficult to use for release note generation. Also, the data size is quite small.

To address the limitations of these three studies, we built a new large-scale dataset called RNSum, which contains approximately 82,000 release notes with commit messages. We reviewed the release notes carefully and redefined the four classes. We also propose classwise summarization methods to automatic release note generation, which can be applied to all English repositories on GitHub.

**Classwise Summarization** There are several reported studies that use class information for summarization. Cao et al. (2017) and Yang et al. (2018) proposed using text categories to improve text summarization. Liang et al. (2019) proposed a clinical note extractive summarization system that generates summaries based on specific disease names. In contrast to these studies, we developed classwise summarization methods for release note generation, which we confirm through experiments to be more effective than the baselines.

## 4 RNSum Dataset

### 4.1 Dataset Construction

We collected the release notes and their associated commit messages from several repositories on GitHub using the GitHub API.[3]

**Repositories** First, we selected all public repositories that did not fork any repositories. A repository that did not fork means a repository that was not copied from others. Then, we filtered out repositories with less than 50 stars, assuming that repositories with many stars tend to contain high-quality release notes, which are suitable for learning reliable release note generation. These filtering steps resulted in 337,048 repositories as of March 2021.

**Release Notes with Classes** We listed the past *releases* for each repository. For each of the four predefined classes, we manually created a vocabulary with up to 30 entries. For example, the vocabulary for the Improvements class contained terms such as "improvements", "enhancements", and "optimizations." We show the vocabularies used in this work in the Appendix. Then, for each release, we searched for the presence of terms in the vocabularies over the entire body text (including the subtitles). We retained the release notes in which at least one class-relevant term was detected in the body text. We removed the repositories where only a single release note class appeared throughout themselves.

**Commit Messages** On GitHub, the release notes are NOT tied to their corresponding commit messages. Therefore, we synchronized the release notes and commit messages using *version tags* (strings) and heuristic matching rules. Specifically, we first listed the version tags (e.g., `v3.7.1`, `v2.6.0`) of the release notes in a repository. Then, the tags with beta versions, such as rc, alpha, and beta, were removed, and the tags were sorted chronologically. Next, considering all adjacent tag pairs, we retained only those that satisfied the heuristic matching rules. The heuristic matching rules focused on only the numerical parts of the version tags (e.g., `v3.7.1` → "371") and compared

[1] https://github.com/nlab-mpg/RNSum-Dataset. For licensing reasons, RNSum does not contain textual content of the release notes and the commit messages but only their URLs. To enable users to obtain the contents easily, we provide scripts using the GitHub API.

[2] The Improvements class includes improvements to the existing features instead of the addition of new features.

[3] https://api.github.com

| | |
|---|---|
| # repositories | 7,216 |
| # release notes (RN) | 81,996 |
| Avg. # release note sentences per RN | 3.3 |
| Avg. # commit messages per RN | 14.9 |
| Avg. # release notes tokens per RN | 63.3 |
| Avg. # commit message tokens per RN | 260.4 |
| Total # unique tokens | 833,984 |

Table 2: Statistics of the RNSum dataset.

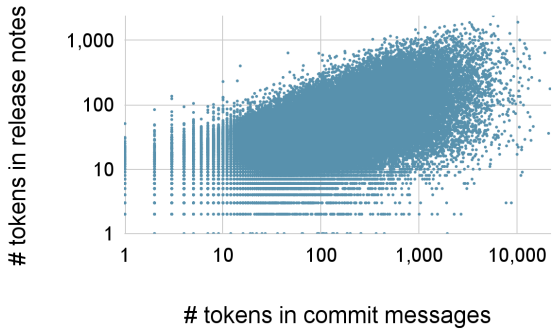


Figure 2: Relationship between the release notes and corresponding commit messages in terms of the numbers of tokens.

the number of digits and the magnitude of the number.[4] For example, given a chronologically sorted version tags, `v3.6.3.1` → `v3.6.4` → `v2.6.1` → `v3.7.0` → `v3.8.0`, only one tag pair, namely `v3.7.0` → `v3.8.0`, can be retained. Then, for each retained tag pair, $(v_{t-1}, v_t)$, we compared the two versions using the GitHub API and collected the list of commit messages for the version tag (or release note) $v_t$.

**Postprocessing** We filtered out release notes that were too complex, with more than 50 sentences and more than 250 commit messages, because it was computationally difficult to handle such large data. Finally, we obtained a total of 7,216 repositories and 81,996 release notes.

### 4.2 Dataset Analysis

We investigate the statistics of the RNsum dataset. Table 2 summarizes the results. Our dataset consists of 81,996 release notes from 7,216 repositories in total. The average number of release note sentences and commit messages per release note is 3.3 and 14.9, respectively. The average number of tokens in release notes and commit messages are 63.3 and 260.4, respectively.[5] The number of unique word types (i.e., vocabulary size) is 833,984, which is significantly large because many project-specific terms such as class and method names are detected.

We plot each data point in RNSum in Figure 2, where each point is represented by a two-dimensional vector of the number of tokens in the release notes and the commit messages. There is a correlation between the release notes and the corresponding commit messages regarding the number of tokens. Also, RNSum contains a wide variety of data of diverse sizes.

We also examine the word overlap rate of the commit messages against the release notes. We remove special symbols such as URL, hash values, and issue numbers by using the spaCy POS tagger.[6] The resulting overlap rate is 56.7%, indicating that extractive approaches (e.g., Glyph), which simply classify commit messages into a fixed set of predefined classes, has a limitation to achieving higher recall.[7] The result also indicates that information outside the commit messages (e.g., pull requests, issues associated with the commit messages) may improve the performance further, which we leave left for future work.

In the end, we examine the distributions of release note classes. There is an obvious class imbalance: Bug Fixes accounts for 60.0%, while Deprecations+ accounts for only 4.2%. This class imbalance problem makes the task more challenging.

## 5 Proposed Methods

Automatic release note generation can be viewed as a task of summarizing commit messages $\boldsymbol{x}$ into the labeled release notes $\boldsymbol{y}_c$. In this paper, we introduce Classwise Extractive-then-Abstractive Summarization (CEAS) and Classwise Abstractive Summarization (CAS) models, which we instantiate by modern transformer-based sequence-to-sequence (seq2seq) networks and can be universally used in various repositories without any special constraints.

[4]The details of the heuristic rules are described in the Appendix.

[5]We used spaCy to tokenize the release notes and the commit messages.

[6]We remove tokens with the following POS tags: PUNCT, PROPN, SYM, NUM, SPCAE, and X.

[7]Therefore, as described in Section 5, our classwise *extractive* summarization method uses a generation model to transform the extracted commit messages into release notes.

### 5.1 Classwise Extractive-then-Abstractive Summarization

The Classwise Extractive-then-Abstractive Summarization (CEAS) model consists of two neural modules: a classifier $F$ and a generator $G$. First, CEAS uses $F$ to classify each commit message into five release-note classes: Features, Improvements, Bug Fixes, Deprecations+, and Other. Then, commit messages classified as the same class are concatenated to form a single document. The commit messages classified as Other are discarded. Then, CEAS applies $G$ to the four labeled documents independently and generates release notes for each class.

In this task, the direct correspondences between commit messages and release notes are not known. Therefore, to train the classifier $F$, we assign pseudo labels to each input commit message using the first ten characters of each commit message. The detail of assigning pseudo labels is described in the Appendix. If pseudo labeling generates commit messages whose class does not appear in the gold release notes, we omit such examples in training. For example, in Figure 1, if the pseudo labeling generates commit messages of Features, the commit messages are discarded because the class Features does not appear in the gold release notes.

### 5.2 Classwise Abstractive Summarization

We model the Classwise Abstractive Summarization (CAS) approach by two different methods. The first model, which we call **CAS-Single**, consists of a single seq2seq network and generates a single long release note text given a concatenation of input commit messages. The output text can be divided into classwise segments based on special class-specific endpoint symbols, like "<Features>" and "</Features>." In training, we concatenate all the gold labeled release notes $\boldsymbol{y}$ into one long document by inserting the classwise endpoint symbols and train the network to generate the target text.

The second method, which we call **CAS-Multi**, consists of four different seq2seq networks $G_c$, each of which corresponds to one of the release-note classes (Features, Improvements, etc.). We train each network $G_c$ to generate the corresponding release notes $\boldsymbol{y}_c$ independently given a concatenation of the input commit messages.

## 6 Experimental Setup

### 6.1 Data

We divided the RNSum dataset into training, validation, and test splits, each containing 74K, 4K, and 4K examples. To avoid data leaks, examples derived from the same repository did not belong to multiple splits. We also removed the training examples with release note text (after concatenation) of longer than 500 tokens to shorten the training time.

### 6.2 Evaluation Metrics

Since a release note $\boldsymbol{y}_c$ can consist of multiple sentences, we concatenate the sentences by inserting spaces and represent the release note as one long text in evaluation. Following the conventional summarization literature, we employ ROUGE (Lin, 2004) as the automatic evaluation metric. We also employ BLEU (Papineni et al., 2002) to evaluate the fluency of generated release notes. Specifically, we compute ROUGE-L (F1), BLEU-3, and BLEU-4 scores. [8] We skip a test example if the reference text is empty. It is also important for the system not to generate release notes when the reference release note is empty (i.e., $\boldsymbol{y}_c = \emptyset$). To evaluate such ability, we also compute *Specificity*, i.e., $\frac{\mathrm{TN}}{\mathrm{TN+FP}}$, where *positive* means that the generated release note is NOT empty.

### 6.3 Baselines

As baseline systems for comparison, we develop Glyph (Pokorný, 2020) and a clustering-based commit message classifier. These baselines are extractive summarization methods because these methods generate release notes by just classifying each input commit message into a fixed set of release-note classes. In contrast, CEAS and CAS employ seq2seq generators to transform input commit messages into novel texts.

**Glyph** Glyph is a publicly available commit message classifier, which groups each input commit message into the following five classes: Features, Improvements, Bug Fixes, Non-functional, and Other. The text classification model relies simply on pretrained word embeddings in fastText (Joulin et al., 2017). Since the Non-functional class is not included in our task, we exclude the commit messages classified as Other or Non-functional from

[8] We calculate the BLEU and ROUGE scores using torchtext `https://github.com/pytorch/text` and HuggingFace framework (Lhoest et al., 2021), respectively.

the output. It must be noted here that Glyph cannot generate the Deprecations+ class.

**Clustering** We also develop a clustering-based classifier for this task. This method classifies each input commit message based on the closest cluster centroid using Euclidean distance. First, we train a Continuous Bag-of-Words (CBOW) model (Mikolov et al., 2013) with a window size of 5 on 10 million commit messages collected from GitHub and obtained 300-dimensional word embeddings for this domain.[9] Then, we embed each input commit message using the averaged embeddings of the first three tokens (without punctuations).[10] Then, we perform the K-means clustering algorithm on the commit message embeddings in the RNSum training set and obtain $k(> 4)$ clusters. We determine the correspondence between the cluster IDs and the release-note classes (Features, Bug Fixes, Improvements, Deprecations+) based on the best alignment $m$ that maximizes the total BLEU scores on the RNSum training set $\mathcal{D}$, i.e.,

$$m^* = \underset{m \in {}_k\mathrm{P}_4}{\operatorname{argmax}} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{D}} \sum_{c \in \mathcal{C}} \mathrm{BLEU}(f_{m(c)}(\boldsymbol{x}), \boldsymbol{y}_c)$$

where $m(c)$ denotes a cluster ID corresponding to the release-note class $c$, and $f_k(\boldsymbol{x})$ is the commit messages classified as the cluster $k$. In inference, input commit messages classified as the remaining $k-4$ clusters are removed in the output. To determine the optimal number of clusters $k$, we tested $k \in [5, 20]$ and found that 11 provided the best validation score.

### 6.4 Implementation Details

**CEAS** We employ BERT (Devlin et al., 2019) and CodeBERT (Feng et al., 2020) as the commit message classifier $F$. CodeBERT is a bimodal pre-trained model for programming language and natural language. Specifically, we apply a multilayer perceptron to the CLS embedding of the input. When training the classifier, class-identifiable words, such as "fix:" and "feat:", are removed because they can be too strong class indicators. We describe the detail of removing class-identifiable words in the Appendix. We employ BART (Lewis et al., 2020) as the generator $G$ (or $G_c$). We used the HuggingFace (Wolf et al., 2020) BertTokenizer, AutoTokenizer, and BARTTokenizer for tokenization. The learning rate was set to 4e-5, and we used the AdamW (Loshchilov and Hutter, 2019) optimizer. Mini-batch size was set to 20 for the classifier and 2 for the generator. To mitigate the class imbalance problem, we also used upsampling for the infrequent classes (Features, Improvements, and Deprecations+). We used the validation set to perform early stopping with a patience of 3 epochs.

**CAS** We employ BART as the seq2seq network. All the CAS-Single and CAS-Multi networks are initialized with the same pretrained parameters, but the parameters are untied across the models and trained independently. We used the HuggingFace (Wolf et al., 2020) BARTTokenizer for tokenization. Mini-batch size was set to 2 for the CAS-Single network and 8 for each network in CAS-Multi. Other training settings are the same as CEAS.

| Method | ROUGE-L (F1) | BLEU-3 | BLEU-4 | Specificity |
|---|---|---|---|---|
| RNSum Test | | | | |
| Glyph | 12.4 | 3.9 | 3.1 | 0.40 |
| Clustering | 10.7 | 4.4 | 3.5 | 0.62 |
| CEAS(BERT) | **26.7** | 4.7 | 4.1 | 0.91 |
| CEAS(CodeBERT) | 25.8 | 4.2 | 3.6 | 0.90 |
| CAS-Single | 23.8 | 3.3 | 2.8 | 0.94 |
| CAS-Multi | 25.5 | **4.8** | **4.2** | **0.98** |
| RNSum Test (Cleaned) | | | | |
| Glyph | 13.7 | 3.0 | 2.3 | 0.39 |
| Clustering | 13.4 | 3.1 | 2.4 | 0.62 |
| CEAS(BERT) | **36.5** | 2.7 | 2.6 | 0.91 |
| CEAS(CodeBERT) | 34.7 | 2.4 | 2.3 | 0.90 |
| CAS-Single | 29.5 | **5.8** | **5.4** | 0.95 |
| CAS-Multi | 35.9 | 3.4 | 3.1 | **0.97** |

Table 3: Results on the RNSum test set. We also show the results on the cleaned version, where we filtered out infrequent tokens like URLs and hash values.

## 7 Results and Discussion

### 7.1 Automatic Evaluation Results

We show automatic evaluation results in Table 3. We also show the results on the cleaned version of the test set, where we removed URLs, hash values, and email addresses, which are significantly difficult to produce accurately.

CEAS and CAS achieved ROUGE-L scores more than 10 points higher than the baselines. In particular, on the cleaned test set, the score gap between the proposed methods and the baselines

[9] We used the gensim library (Řehůřek and Sojka, 2010). We removed words with frequencies lower than 300 occurrences.

[10] We also tested the average embeddings of all tokens or all nouns. The averaging of the first three tokens consistently outperforms these two counterparts.

| | |
|---|---|
| CM | - fix createOptions validation issue (#294) * fix createOptions validation issue * Update logic - 2.5.1 (#295) |
| Class | Bug Fixes |
| PR + Issues | - fix createOptions validation issue https://github.com/Microsoft/vscode-azure-iot-toolkit/pull/294 - Error if create options is larger then 512 bytes https://github.com/Microsoft/vscode-azure-iot-toolkit/issues/293 |
| RN | - Fix deployment JSON validation issue when create options is larger than 512 bytes |

Table 4: An example provided for human evaluators. This example requires references to pull requests (PR) and issues. CM and RN denote commit messages and release notes to be evaluated, respectively.

jumped to more than 20 points. These results indicate that CEAS and CAS are significantly effective. In addition, CEAS got a better ROUGE-L score than CAS, suggesting that combining a classifier and a generator is effective and training the classifier using pseudo labels. The high coverage of CEAS can be achieved probably because the classifier can focus on selecting relevant commit messages for each class. Moreover, CEAS(BERT) got higher scores than CEAS(CodeBERT), indicating that it is better to use BERT for tasks where the commit message is the input. CodeBERT is closer to the domain of commit messages than BERT, but we assume that this is because it was trained with relatively little natural language data. Furthermore, CAS-Multi tended to yield higher ROUGE-L than CAS-Single, suggesting that it is also effective to independently develop different abstractive summarization models for each release-note class. Although not as apparent as ROUGE-L, CAS models (CAS-Single and CAS-Multi) generated comparable or higher BLEU scores than CEAS and the baselines. CAS models were also able to achieve significantly higher Specificity scores (> 30 points) compared to the baselines. These results indicate that CAS models can generate less noisy release notes than the baselines. We hypothesize that CAS is trained a lot to remove noise from all commit messages, including Other class, which strengths the ability to deal with noise.

### 7.2 Human Evaluation Results

We employed twelve human evaluators to manually assess the quality of the release notes generated by the systems and the reference release notes. The evaluators were graduate students or working professionals with at least one year of experience reading release notes and updating software libraries. We randomly chose 120 release notes from the test set. The allocations of the Features, Improvements, Bug Fixes, and Deprecations+ classes were 40, 25, 40, and 15, respectively. We divided the evaluation tasks into three groups of 40 questions, and each group was assigned to four different evaluators. We used a crowdsourcing platform, Yahoo! Crowdsourcing,[11] operated by Yahoo Japan Corporation for the evaluations. In the following, we explain the evaluation task and scoring measures.

**Evaluation Task** For each evaluation task, an evaluator is given a list of input commit messages and the target release note class. The evaluator is also given supplemental information about pull requests and issues. These supplemental data were not used to train the models, but we included them because they are often helpful for accurately evaluating the release notes. We show an example of this case in Table 4. The release notes that were manually prepared by the original maintainers contained the words "JSON" and "larger than 512 bytes", but this information cannot be found in the commit messages. To accurately evaluate the human-generated release note, pull requests and issues are required. Thus, we instructed the evaluators to check the titles, pull requests, and issues if necessary. We selected CAS-multi for its better performance than the CEAS and CAS-single in terms of the quality of generated texts. For the Deprecations+ class, we evaluated the outputs of the CAS-Multi, the Clustering model, and the human reference because the Glyph does not produce release notes for the Deprecations+ class.

**Scoring** We employed a five-point scoring scheme for evaluating the release notes. The evaluation scores were determined based on two criteria: Percentage of necessary information (coverage) and percentage of unnecessary information (noise contamination). For the coverage-oriented scoring, the following guidance was used for the scoring; 5: 90% or more necessary information (NI), 4: 70% or more NI, 3: 50% or more NI, 2: 30% or more NI, and 1: less 30% of NI. For the noise-oriented scoring, the following guidance was used; 5: no unnecessary information (UI), 4: less UI, 3: slightly less UI, 2: a little UI, 1: much UI. We use Fleiss' Kappa to measure inter-annotator

[11] https://crowdsourcing.yahoo.co.jp/.

| | Method | Human Rating (%) 1 | 2 | 3 | 4 | 5 | Avg. |
|---|---|---|---|---|---|---|---|
| Cov. + Noise | Glyph | 28 | 19 | 16 | 16 | 21 | 2.82 |
| | Clustering | 22 | 19 | 19 | 15 | 25 | 3.01 |
| | CAS-Multi | 14 | 11 | 16 | 14 | 45 | **3.64** |
| | Human | 13 | 11 | 13 | 18 | 45 | 3.71 |
| Cov. | Glyph | 22 | 19 | 19 | 16 | 24 | 3.01 |
| | Clustering | 20 | 19 | 22 | 14 | 25 | **3.05** |
| | CAS-Multi | 19 | 18 | 24 | 16 | 23 | **3.05** |
| | Human | 15 | 13 | 16 | 20 | 36 | 3.50 |
| Noise | Glyph | 34 | 20 | 13 | 15 | 18 | 2.64 |
| | Clustering | 25 | 19 | 15 | 16 | 25 | 2.98 |
| | CAS-Multi | 8 | 5 | 8 | 12 | 67 | **4.24** |
| | Human | 11 | 9 | 10 | 16 | 54 | 3.92 |

Table 5: Human evaluation results for the generated or reference release notes. The numbers below the 5-point scale indicate percentages.

agreement (Davies and Fleiss, 1982) and obtain a score of 0.39, which indicates fair agreement.

**Results** We show the results of the human evaluations in Table 5. For all the metrics, CAS-Multi achieves the highest human evaluation scores among the automatic systems (See the column of "Avg."). In particular, in the noise-oriented (or precision-oriented) metric, CAS-Multi significantly improves over the baselines and even outperforms the human references. This fact suggests that the abstractive summarization approach is effective in transforming the noisy textual representations in the commit messages. However, the CAS-Multi's performance is still lower than those of the human references, suggesting the remaining challenges in this task.

We also tested the statistical significance of the results using a permutation test (Pitman, 1937). Since evaluating all possible permutations would require a considerable amount of time, we used the approximation method. Note that sampling all permutation is typically not feasible unless the dataset size is relatively small.[12] We set the number of rounds to 10,000. We applied Cov. + Noise scores to the test. Comparing CAS-Multi with the baselines, all the p-values of the permutation tests were less than 0.001, indicating that the improvements are statistically significant.

[12] http://rasbt.github.io/mlxtend/user_guide/evaluate/permutation_test/.

| |
|---|
| Commit Messages (Improvements):<br>- Add deprecation comments to channels.*, groups.*, im.*<br>- add test for EscapeMessage funcs<br>- missing assertion in bot test<br>- adding support for #845<br>- Add conversations.list to the slacktest server<br><br>Generated Release Note (Cov.: 3.0, Noise: 3.0):<br>- Add deprecation comments to channels.*, groups.*, im.*<br><br>Reference Release Note (Cov.: 4.5, Noise: 3.5):<br>- Add test for EscapeMessage function in slack<br>- add missing result evaluation in bot test<br>- Add conversations.list to the slacktest server<br>- adds support for listing files hidden the free tier file limit |
| Commit Messages (Bug Fixes):<br>- fix date time filter ranges<br>- fix formula parser<br>- fix excel<br><br>Generated Release Note (Cov.: 2.5, Noise: 5.0):<br>- Fixed date time filter ranges<br><br>Reference Release Note (Cov.: 3.2, Noise: 2.7):<br>- Excel export: first column not formatted.<br>- Formula parser issue with comma inside string.<br>- Not properly working Datetime filters Today, On, Between. |
| Commit Messages (Deprecations+):<br>- Drop the parse package in favor of using protokit<br>- Ensure ExcludePatterns is still an option<br><br>Generated Release Note (Cov.: 2.7, Noise: 2.7):<br>- The entire 'parser' package (in favor of [protokit])<br><br>Reference Release Note (Cov.: 4.0, Noise: 3.5):<br>- Dropped the 'ExcludePatterns' package in favor of using protokit for parsing |

Table 6: Three release note examples generated by CAS-Multi. We also show the commit messages and the reference release notes. Cov. and Noise show the human-evaluated coverage and noise scores, respectively.

### 7.3 Error Analysis

We qualitatively analyzed the outputs of CAS-Multi to identify the bottlenecks of the current approach. Table 6 shows several examples where the outputs of CAS-Multi and the human references are largely different. The input commit messages are lengthy, so we only show the commit messages related to each class in this table; the entire text is shown in the Appendix.

First, we found that CAS-Multi tends to produce significantly shorter release notes than human references. In the first two examples in Table 6, CAS-Multi generates only a single sentence, while the human references contain multiple sentences. This is probably due to the fact that the release note classes are significantly scarce in the training

set, and the model is trained to be reluctant to generate release note text. Actually, the numbers of release notes containing the Features and Improvements classes in the training set are only 33.4% and 14.0%, respectively. We used the upsampling technique to reduce the class imbalance problem; however, upsampling cannot inherently increase the number of unique training examples. Therefore, it is necessary to explore ways to augment the training patterns inherently in the future.

Second, we found that release notes are often difficult to precisely and accurately produce without supplemental information outside the commit messages. In the second example in Table 6, CAS-Multi generates just "Fixed date time filter ranges," which is a simple paraphrase of the first commit message. In contrast, the human reference enriches the content by rephrasing it as "Not properly working Datetime filters Today, On, Between." Moreover, the third commit message ("fix excel") is enriched as "Excel report: first column not formatted," which is impossible to generate without external information. In the last example in Table 6, without background knowledge of the repository, it is impossible to detect relationships between the two commit messages and to combine them into a single sentence.

## 8 Conclusion

In this paper, we presented a new large-scale dataset for automatic generation of release notes. The dataset comprised approximately 82k release notes from over 7k repositories on GitHub. We formulated a task to automatically generate release notes by summarizing the commit messages, which can be applied to all software development projects that use English. We confirmed the validity of the proposed classwise extractive-then-abstractive summarization (CEAS) model and the classwise abstractive summarization (CAS) model via experiments to generate less noisy release notes at higher coverage than the baselines. However, there are still gaps in the coverage performance compared to manually generated outputs; it is expected that this could be improved by including additional information such as issues and pull requests with the commit logs.

## References


Ziqiang Cao, Wenjie Li, Sujian Li, and Furu Wei. 2017. Improving multi-document summarization via text classification. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 3053–3059. AAAI Press.

Mark Davies and Joseph L Fleiss. 1982. Measuring agreement for multinomial data. *Biometrics*, pages 1047–1051.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547, Online. Association for Computational Linguistics.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2073–2083, Berlin, Germany. Association for Computational Linguistics.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2017. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 427–431. Association for Computational Linguistics.

Sebastian Klepper, Stephan Krusche, and Bernd Bruegge. 2016. Semi-automatic generation of audience-specific release notes. In *Proceedings of the International Workshop on Continuous Software Evolution and Delivery*, CSED '16, page 19–22, New York, NY, USA. Association for Computing Machinery.

Stanislav Levin and Amiram Yehudai. 2017. Boosting automatic commit classification into maintenance activities by utilizing source code changes. In *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering*, PROMISE, page 97–106, New York, NY, USA. Association for Computing Machinery.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*,

pages 7871–7880, Online. Association for Computational Linguistics.

Quentin Lhoest, Patrick von Platen, Thomas Wolf, Albert Villanova del Moral, Yacine Jernite, Suraj Patil, Mariama Drame, Julien Chaumond, Julien Plu, Joe Davison, Simon Brandeis, Teven Le Scao, Victor Sanh, Kevin Canwen Xu, Lewis Tunstall, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Lysandre Debut, Clément Delangue, Théo Matussière, Stas Bekman, and François Lagunas. 2021. huggingface/datasets: 1.7.0.

Jennifer Liang, Ching-Huei Tsou, and Ananya Poddar. 2019. A novel system for extractive clinical note summarization using EHR data. In *Proceedings of the 2nd Clinical Natural Language Processing Workshop*, pages 46–54, Minneapolis, Minnesota, USA. Association for Computational Linguistics.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Andreas Mauczka, Florian Brosch, Christian Schanes, and Thomas Grechenig. 2015. Dataset of developer-labeled commit messages. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, MSR '15, page 490–493. IEEE Press.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, page 3111–3119, Red Hook, NY, USA. Curran Associates Inc.

Laura Moreno, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Andrian Marcus, and Gerardo Canfora. 2014. Automatic generation of release notes. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, page 484–495, New York, NY, USA. Association for Computing Machinery.

Sheena Panthaplackel, Pengyu Nie, Milos Gligoric, Junyi Jessy Li, and Raymond Mooney. 2020. Learning to update natural language comments based on code changes. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1853–1868, Online. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Edwin JG Pitman. 1937. Significance tests which may be applied to samples from any populations. *Supplement to the Journal of the Royal Statistical Society*, 4(1):119–130.

Fridolín Pokorný. 2020. Glyph. https://github.com/thoth-station/glyph.

Nasir Safdari. 2018. Multi-class-text-classification—-random-forest. https://github.com/nxs5899/Multi-Class-Text-Classification----Random-Forest.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Min Yang, Qiang Qu, Ying Shen, Qiao Liu, Wei Zhao, and Jia Zhu. 2018. Aspect and sentiment aware abstractive review summarization. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1110–1120, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Pengcheng Yin and Graham Neubig. 2018. TRANX: A transition-based neural abstract syntax parser for semantic parsing and code generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 7–12, Brussels, Belgium. Association for Computational Linguistics.

Radim Řehůřek and Petr Sojka. 2010. Software framework for topic modelling with large corpora. In *Proceedings of LREC 2010 workshop New Challenges for NLP Frameworks*, pages 46–50, Valletta, Malta. University of Malta.

# Appendices

## A Dataset Collection Details

**Vocabularies for Detecting Label** When we collected the release notes, we divided them into four labels: Features, Improvements, Bug Fixes, Deprecations, Removals, and Breaking Changes. Since label names vary from a repository to repository, e.g., a label Improvements has improved, improvements, optimizations, etc., we set up candidates for appropriate label names for each label in advance. Table 7 shows the list of vocabularies for each label used to identify the labels in the release notes.

**Heuristic Rules for Extracting Commit Messages** We listed the version tags (e.g., `v3.7.1`, `v2.6.0`) of the release notes for extracting corresponding commit messages. The heuristic matching rules focus only on the numeric part of the version tags (e.g., "371", "260") and compare the number of digits and the size of the number. Let us define "base" as the past tag and "head" as the future tag. We do not include the tag pairs "base" and "head" in our dataset if any of the following three cases apply. (1) The number of digits in "base" and "head" are different. (2) "base" is greater than "head". (3) One or more of the values of each digit of "head" is greater than the value of the corresponding digit of "base" + 1. For example, given the version tags, `v3.6.3.1` → `v3.6.4` → `v2.6.1` → `v3.7.0` → `v3.8.0`, we can obtain one version tag pair, `v3.7.0` → `v3.8.0`. It is because `v3.6.3.1` → `v3.6.4` is rejected with a constraint of (1), `v3.6.4` → `v2.6.1` is rejected with a constraint of (2), and `v2.6.1` → `v3.7.0` is rejected with a constraint of (3).

| Label | Vocabularies |
|---|---|
| Features | new, add, adds, feat, feats, added, adding, feature, addings, features, featured, addition, additions, new stuff, additional, additionals, new feature, new features, additionally, features added |
| Improvements | improve, enhance, improved, improves, optimize, optimise, enhanced, optimizes, optimized, improvement, enhancement, improvements, enhancements, optimisation, optimization, optimisations, optimizations, other improvements, security improvement, security improvements, performance improvement, implemented enhancement, performance improvements, implemented enhancements |
| Bug Fixes | bug, fix, bugs, fixed, fixes, patch, fixbug, bugfix, hotfix, modify, fixing, patches, bug fix, bigfixes, bugfixes, hotfixes, modified, bugfixing, bug fixes, fixed bugs, bugs fixed, bugs fixes, critical bug, resolved bugs, security fixes, bug fix release, major bug fixes, fix several bugs, notable bug fixes, security bugfixes |
| Deprecations+ | delete, remove, deleted, removed, removal, removes, removals, breaking, deletion, deprecate, deprecated, deprecation, deprecations, compatibility, incompatibility, breaking change, breaking changes, deprecation warning, deprecation warnings, potentially breaking, removed deprecations, deprecations and removed, for future compatibility, deprecations and removals, backward incompatible changes, backwards incompatible changes, breaking changes and deprecations, new deprecations |

Table 7: List of vocabularies for each label used to identify the labels in the release notes.

## B CAS Preprocessing Details

**Pseudo labels used for commit message classification** Psuedo labels are made up of Features, Improvements, Bug Fixes, Deprecations+, and Other. Since type prefix names vary from a repository to repository, e.g., a label Improvements has improve, perf, optimize, etc., we set up candidates for appropriate type prefix names for each label in advance. Table 8 shows the list of type prefixes for each label used to identify the pseudo labels in the commit messages. We read the first ten characters of each commit message, and if a word in the type prefix dictionary existed, we assigned a corresponding pseudo label to it. The total number of commit messages for the training data assigned pseudo labels was about 460,000, which was 42% of the total commit messages. The numbers for each class of Features, Improvements, Bug Fixes, Deprecations+, and Other are 129,580, 9,672, 146,233, 26,733, and 149,994.

**Removal of class-identifiable words** We remove words that can be used as class-identifiable markers from commit messages to train the classification model. The common class-identifiable word is a type name followed by a colon, like feat: and fix:. We also consider parentheses, brackets, and asterisk forms to the colon. Therefore, in the case of feat, we regarded feat:, (feat), [feat], **feat** as type prefixes and removed them from commit messages.

| Psuedo Label | Type prefix |
|---|---|
| Features | feat, add, new, feature |
| Improvements | improve, perf, optimize |
| Bug Fixes | fix, bug, bugfix |
| Deprecations+ | breaking, deprecate, remove |
| Other | chore, refactor, test, doc, build, style, ci, revert |

Table 8: List of type prefixes for each label used to identify the pseudo labels in the commit messages.

## C Qualitative Results with Entire Text

| |
|---|
| Input Commit Messages:<br>- Add deprecation comments to channels.*, groups.*, im.* Add deprecation comment for deprecated APIs in Slack. We decided to announce to users in advance before removing these methods. See: https://api.slack.com/changelog/2020-01-deprecating-antecedents-to-the-conversations-api<br>- add ommitempty json tag to text field<br>- remove empty field "text":"" from tests in chat_test.go, attachments_test.go<br>- Use custom ctx on SetUserCustomStatusContext<br>- add test for EscapeMessage funcs<br>- Merge pull request #833 from sryoya/add_test_for_escapte_message_func Add test for EscapeMessage function in slack<br>- missing assertion in bot test<br>- Add configuration of golangci-lint We are planning to migrate from gometalinter to golangci-lint. Once we complete the migration from Travis CI to GitHub Actions, we remove .gometalinter.json.<br>- Use GitHub Actions for Go 1.12+<br>- Fix existing linter issues<br>- reminders time field fix<br>- adding support for #845<br>- Add conversations.list to the slacktest server This commit adds the ‘conversations.list‘ endpoint to the testing server. This endpoint returns JSON shaped the same as ‘channels.list‘, so I just reused that handler. I also found what appeared to be a bug with error handling (a misnamed error) so I fixed that as well.<br>- Merge pull request #840 from slack-go/github-actions Use golangci-lint and GitHub Actions<br>- Fix unintended json marshalling error resolve: #851<br>- Merge pull request #815 from slack-go/deprecated-apis Add deprecation comments to channels.*, groups.*, im.*<br>- Remove conf for travis<br>- Merge pull request #855 from slack-go/remove-travis Remove configuration for Travis CI<br>- Merge pull request #830 from GLOFonseca/GLOFonseca-add-custom-ctx-SetUserCustomStatusContext Use custom ctx on SetUserCustomStatusContext<br>- Merge pull request #842 from sryoya/missing_assertion_in_bot_test add missing result evaluation in bot test<br>- Merge pull request #853 from hobbeswalsh/add_conversations_list Add conversations.list to the slacktest server<br>- Merge pull request #846 from rk295/rk/845 adds support for listing files hidden by the free tier file limit<br>- Merge pull request #843 from KarolisKl/reminders_time_fix reminders time field fix<br>- Merge pull request #788 from prgres/attachment-text-field-json-tag#784 Resolve: Revert changes to json tags in Attachment’s Text field<br>- Merge pull request #854 from slack-go/fix-851 Fix unintended json marshalling error |
| Glyph Release Note (Cov.: 3.2, Noise: 2.2):<br>- add test for EscapeMessage funcs<br>- missing assertion in bot test<br>- Merge pull request #842 from sryoya/missing_assertion_in_bot_test add missing result evaluation in bot test<br>- Merge pull request #853 from hobbeswalsh/add_conversations_list Add conversations.list to the slacktest server |
| Clutering Release Note (Cov.: 2.5, Noise: 2.7):<br>- missing assertion in bot test<br>- Add configuration of golangci-lint We are planning to migrate from gometalinter to golangci-lint. Once we complete the migration from Travis CI to GitHub Actions, we remove .gometalinter.json.<br>- reminders time field fix<br>- adding support for #845<br>- Add conversations.list to the slacktest server This commit adds the ‘conversations.list‘ endpoint to the testing server. This endpoint returns JSON shaped the same as ‘channels.list‘, so I just reused that handler. I also found what appeared to be a bug with error handling (a misnamed error) so I fixed that as well. |
| CEAS Release Note:<br>empty |
| CAS-Multi Release Note (Cov.: 3.0, Noise: 3.0):<br>- Add deprecation comments to channels.*, groups.*, im.* |
| Reference Release Note (Cov.: 4.5, Noise: 3.5):<br>- Add test for EscapeMessage function in slack<br>- add missing result evaluation in bot test<br>- Add conversations.list to the slacktest server<br>- adds support for listing files hidden the free tier file limit |

Table 9: A release note example generated by three models in Improvements label. We also show the input commit messages and the reference release notes. Cov. and Noise show the human-evaluated coverage and noise scores, respectively.

| |
|---|
| Input Commit Messages:<br>- fix date time filter ranges<br>- calendar dashlet view link<br>- panel tpl fix<br>- calendar dashlet month title<br>- installer: added possibility to load values from existing data/config.php<br>- installer: load values from existing data/config.php<br>- Changed news URL<br>- lang portal permission fix<br>- Fixed undefined Log class<br>- Merge branch 'hotfix/4.8.2' of ssh://172.20.0.1/var/git/espo/backend into hotfix/4.8.2<br>- bottom loaded with middle<br>- version<br>- fix lang<br>- fix lang<br>- fix empty varchar and text<br>- layout noLabel param<br>- fix recotd stop listen window<br>- fix formula parser<br>- fix excel<br>- cleanup attachment improvements<br>- remove attachment if file/image changed<br>- fix typo<br>- email inline attachment cleanable<br>- it_IT lang fixes<br>- fix application set |
| Glyph and Clustering Release Note (Cov.: 3.0, Noise: 2.5):<br>- fix date time filter ranges<br>- fix lang<br>- fix empty varchar and text<br>- fix recotd stop listen window<br>- fix formula parser<br>- fix excel<br>- fix typo<br>- fix application set |
| CEAS:<br>- Calendar dashlet next/previous buttons and some css fixes.<br>- Fixed a bug where the date and time filter were not being applied properly. |
| CAS-Multi Release Note (Cov.: 2.5, Noise: 5.0):<br>- fix date time filter ranges |
| Reference Release Note (Cov.: 3.2, Noise: 2.7):<br>- Excel export: first column not formatted.<br>- Formula parser issue with comma inside string.<br>- Not properly working Datetime filters Today, On, Between. |

Table 10: A release note example generated by three models in Bug Fixes label. We also show the input commit messages and the reference release notes. Cov. and Noise show the human-evaluated coverage and noise scores, respectively.

Input Commit Messages:

- Add support for -help, -h, and -version flags
- Add more details to help text
- Merge pull request #321 from pseudomuto/add_help_and_version Add support for -help, -h, and -version flags
- nobr: preserve blank lines
- Merge pull request #323 from meteor/glasser/blank-lines nobr: preserve blank lines
- Remove some unused code
- Support exclude file patterns
- Update docs and tests
- fix wrong command example md is not build-in format. markdown is correct
- Add IgnoreMe proto and update make examples/docker_test. - Use file.GetName() instead of string deref
- Merge pull request #326 from notbdu/master Support file exclude patterns
- Merge pull request #328 from suusan2go/patch-1 fix wrong command example
- Be a little clearer about invocation [ci skip]
- dep init, retool, protoc 3.5.1, and go 1.10
- Update CHANGELOG with unreleased things
- Wrong version for compare [ci skip]
- Merge pull request #344 from pseudomuto/migrate_t_dep_and_retool Upgrade a few tools
- Use protokit Plugin interface for the generator
- Drop the parse package in favor of using protokit
- Ensure ExcludePatterns is still an option
- Add -Iprotos to entrypoint to keep file names the same
- Merge pull request #347 from pseudomuto/use_protokit_for_parsing Use protokit for parsing
- Simplify fixtures
- Use Make for generating resources
- Clean up the Makefile a bit
- Move a few things and add some badges
- Merge pull request #348 from pseudomuto/cleanup_dev_test_things Clean up dev test things
- Move resources app into resources
- Update go get package [ci skip]
- Update CHANGELOG.md [ci skip]
- Bump version to 1.1.0
- Bump version to 1.1.0

Clustering Release Note (Cov.: 1.5, Noise: 3.0):

- Remove some unused code

CEAS:

- Removed some unused code.

CAS-Multi Release Note (Cov.: 2.7, Noise: 2.7):

- The entire ‘parser‘ package (in favor of [protokit])

Reference Release Note (Cov.: 4.0, Noise: 3.5):

- Dropped the ‘ExcludePatterns‘ package in favor of using protokit for parsing

Table 11: A release note example generated by two models in Deprecations+ label. We also show the input commit messages and the reference release notes. Cov. and Noise show the human-evaluated coverage and noise scores, respectively.