ReleaseEval: A Benchmark for Evaluating Language Models in Automated Release Note Generation

Qianru Meng and Zhaochun Ren and Joost Visser
The Leiden Institute of Advanced Computer Science (LIACS), Leiden University
{q.r.meng, z.ren, j.m.w.visser}@liacs.leidenuniv.nl

Abstract

Automated release note generation aims to reduce the cost of documenting frequent software updates. Existing benchmarks mainly rely on commit messages, limiting evaluation of how models use broader software-change evidence. We introduce RELEASEEVAL, a reproducible and openly licensed multi-granularity benchmark with 94,987 release notes from 3,369 repositories across six programming languages. Each release is aligned with commit messages, commit tree structures, and raw code diffs, supporting three task settings: *commit2sum*, *tree2sum*, and *diff2sum*. Automatic and human evaluations show that models benefit from lightweight commit-tree structure, while raw diffs remain challenging due to length and noise. These findings position RELEASEEVAL as a multi-granularity benchmark for studying release note generation. We release the dataset and code at https://github.com/Changanyuexia/rn_benchmark.

1 Introduction

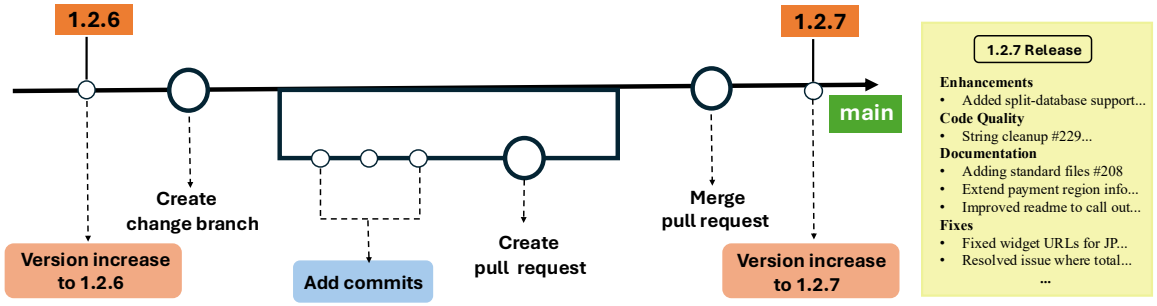
Release notes are key documents in software projects, summarizing changes, enhancements, and fixes across versions (Alali et al., 2008; Yu, 2009). As systems evolve rapidly, manual documentation becomes costly and error-prone, motivating research on automated release note generation. Early work focused on extractive methods (Moratanch and Chitrakala, 2017), while later studies adopted neural abstractive approaches (Jiang et al., 2021; Kamezawa et al., 2022). More recently, large language models (LLMs) have further advanced this task by offering greater fluency and contextual understanding (Daneshyan et al., 2025).

Despite this progress, existing benchmark datasets (Nath and Roy, 2022; Jiang et al., 2021; Kamezawa et al., 2022; Daneshyan et al., 2025) for release note generation suffer from two major shortcomings that limit their research and practical value. First, licensing and reproducibility remain unresolved. Although many are open-sourced, the absence of explicit distribution licenses prevents their legal reuse and redistribution. Moreover, some datasets have limited reproducibility. Even when the repositories used in data collection are documented, these repositories evolve over time, leaving parts of the datasets unavailable (Kamezawa et al., 2022). Second, dataset design directly affects the quality of release note generation. Most existing datasets define the task narrowly as commit-to-release generation, thereby overlooking critical information in the structural hierarchy of commits and fine-grained code changes. As illustrated in Figure 1(b), software commits typically contain not only commit messages but also a branching structure with parent–child relationships, which we refer to as the commit tree. Furthermore, as shown in Figure 1(c), code diffs provide precise, line-level details essential for understanding the nature and impact of changes.

To address these gaps, we introduce RELEASEEVAL, an open-access and reproducible benchmark for multi-granularity release note generation. RELEASEEVAL contains 94,987 release notes from 3,369 repositories across six programming languages, and aligns each release with three levels of software-change evidence: commit messages, commit tree structures, and raw code diffs. Based on this alignment, we organize release note generation into three task settings, *commit2sum*, *tree2sum*, and *diff2sum*, which correspond to semantic, structural, and code-level evidence.

Rather than treating these tasks as a simple ranking of input formats, RELEASEEVAL uses them to examine how language models behave under different levels of software-change abstraction. Our evaluation shows that LLMs can effectively use developer-written commit information and benefit from lightweight commit-tree structure. At the same time, raw code diffs expose a more challeng-

Findings of the Association for Computational Linguistics: IJCNLP-AACL 2026, pages 2514–2526
November 6-10, 2026 ©2026 Association for Computational Linguistics



(a) A release pipeline of amazon-payments-magento-2-plugin for version 1.2.6 to v1.2.7

Structure	Hash	Commit messages
* *	acd0d03c	Version increase to 1.2.7
* *	c0115f78	Update to 1.x for APS-24 (#300)
...	...	...
| \		
| * *	2169f3d5	Update - removed reference to Mail class in 1.2.6 di.xml file
* | | *	cbf82502	Fix widget URLs for JP region
* | / / *	9d5e9765	Added missing line for list formatting
| /		
...	...	...
* *	5402c9d7	Merge pull request #204 from lemked/pre_1.2.6
* *	386e0a11	Increase version to 1.2.6

(b) A commit tree example for version 1.2.6 to 1.2.7

```
Commit: Update to 1.x for APS-24 (#300)
File path: src/Login/Controller/Login.php
 + namespace Amazon\Login\Controller;
 - use Amazon\Core\Domain\AmazonCustomer;
File path: src/Login/Controller/Login/Guest.php
 + private function getRedirectLogin() {
        return $this->_redirect($this->customerUrl
>getLoginUrl());} ...
Commit: Merge pull request #204 from lemked/pre_1.2.6
File path:  src/Payment/etc/di.xml
 - <type name="Magento\Framework\Mail\MessageInterface"
shared="false" />
   <type name="Amazon\Payment\Model\Config">
   ...
```

(c) Two examples of code diff for version 1.2.6 to 1.2.7

Figure 1: The details of commits from Amazon Payments Magento 2 plugin. Figure (a): a release pipeline which includes branch-level commits. Figure (b): commit tree, which includes commit messages and their structures. Figure (c): two fine-grained code diffs, corresponding to the messages. Note: (a) and (b) highlight matched commits: blue denotes commits added on the change branch, orange indicates version-bump commits on the main branch. In (c), red marks the commits corresponding to (b), green highlights added code, and pink marks deleted code.

ing setting: although they provide fine-grained implementation evidence, their length and low-level nature make release-level abstraction difficult. Beyond automatic metrics, expert human evaluation further assesses both benchmark quality and generated release notes from the perspectives of completeness, clarity, conciseness, and organisation. Together, these results position RELEASEEVAL as a benchmark for both release note generation and long-context software-change understanding.

Our contributions are summarized as follows:

- We present RELEASEEVAL, a large-scale, reproducible, and license-aware benchmark for automated release note generation, covering 94,987 release notes from 3,369 repositories across six programming languages.
- We organize release note generation into three task settings, *commit2sum*, *tree2sum*, and *diff2sum*, enabling systematic evaluation across semantic, structural, and code-level evidence.
- We conduct expert human evaluation to assess both benchmark quality and generated release notes.
- We evaluate representative language models under fine-tuning and few-shot settings, revealing how current models exploit different levels of software-change evidence.

2 Related Work

Methods. Prior work on automated release note generation mainly follows extractive or abstractive summarization paradigms. Extractive methods select salient sentences or segments from source artifacts such as pull requests, change logs, or commit messages. Early systems include the semi-automated approach of Klepper et al. (2016) and the ARENA tool proposed by Moreno et al. (2016). Later work adopts embedding-based ranking methods; for example, Nath and Roy (2022) combine GloVe embeddings (Pennington et al., 2014) with TextRank (Mihalcea and Tarau, 2004) to identify important commits for release notes. Abstractive methods instead generate release notes from change-related inputs. Jiang et al. (Jiang et al., 2021) propose DeepRelease, which uses an LSTM-based encoder–decoder model for release note generation. RNSum (Kamezawa et al., 2022) follows a classwise abstractive summarization paradigm,

Features	Nath (Nath and Roy, 2022)	DeepRelease (Jiang et al., 2021)	RNSum (Kamezawa et al., 2022)	SmartNote (Daneshyan et al., 2025)	ReleaseEval
Release Notes	1,213	37,000	81,996	21,882	94,987
Repositories	13	561	7,216	272	3,369
Reproducible	✓	✓	Semi	✓	✓
Deduplicated	✓	✓	✓	✓	✓
License-Aware Data Collection	✗	✗	✗	✗	✓
Additional Filtering	✓	✓	✓	✓	✓
Programming Languages	3	8	NA	7	6

Table 1: Comparison between RELEASEEVAL and existing datasets for release note generation.

while SmartNote (Daneshyan et al., 2025) combines commit classification with LLM-based generation. These studies show the feasibility of automatic release note generation, but most model the task mainly from commit-level textual descriptions.

Datasets. Dataset construction remains a central challenge for release note generation. Nath et al. (Nath and Roy, 2022) built a dataset of 1,200 GitHub releases paired with commit messages and PR titles. DeepRelease (Jiang et al., 2021) collected over 46K release notes associated with pull requests. RNSum (Kamezawa et al., 2022) introduced a larger dataset of around 82K release notes derived from commit messages, and SmartNote (Daneshyan et al., 2025) released over 21K release notes aligned with commit messages. As summarized in Table 1, these datasets have advanced the field, but they still exhibit limitations in licensing, reproducibility, or input coverage. In particular, existing benchmarks largely constrain release note generation to commit-message-based inputs, leaving structural commit histories and fine-grained code changes underexplored. RELEASEEVAL addresses this gap by providing a license-aware, reproducible benchmark that aligns release notes with commit messages, commit tree structures, and raw code diffs, enabling evaluation across multiple granularities of software-change evidence.

3 ReleaseEval

3.1 Overall Framework

The construction of RELEASEEVAL follows a structured framework encompassing data collection and processing, benchmark construction and evaluation, as illustrated in Figure 2. The following section provides a detailed explanation of each component.

3.2 Data Collection and Processing

3.2.1 Project Selection

The projects used in our dataset are sourced from CommitChronicle (Eliseeva et al., 2023), a benchmark for commit message generation that provides openly licensed data from over 12,000 repositories. To ensure project quality and activity, we first filter for repositories with more than 50 stars and at least 10 contributors. However, as Hu et al. (2016) point out, starred projects do not necessarily represent the most important or actively maintained ones. Therefore, we further refine our selection by retaining only repositories with an issue resolution rate above 40% and with recent commit activity between August 1, 2017, and October 5, 2024. To ensure diversity, we also analyze repository ownership and find that 86% belong to organizations while 14% are maintained by individual developers. Additionally, we focus on repositories implemented in six of the most widely used programming languages on GitHub—TypeScript, JavaScript, Go, Python, Java, and PHP—selected from an initial pool of 20 programming languages. After applying all filters, we obtain a final selection of 3,369 projects from the original 12.4K repositories.

3.2.2 Dataset Collection

After selecting the target projects, we systematically collected a comprehensive set of attributes, summarized in Table 6 (see in Appendix A), that capture the metadata associated with releases, commits, and code changes. To ensure scalability and reproducibility, we implemented an automated data collection pipeline leveraging the GitHub API (GitHub, 2025) for retrieving release notes and commit trees, and employed PyDriller (Spadini et al., 2018) to extract commit metadata and fine-grained code diffs.

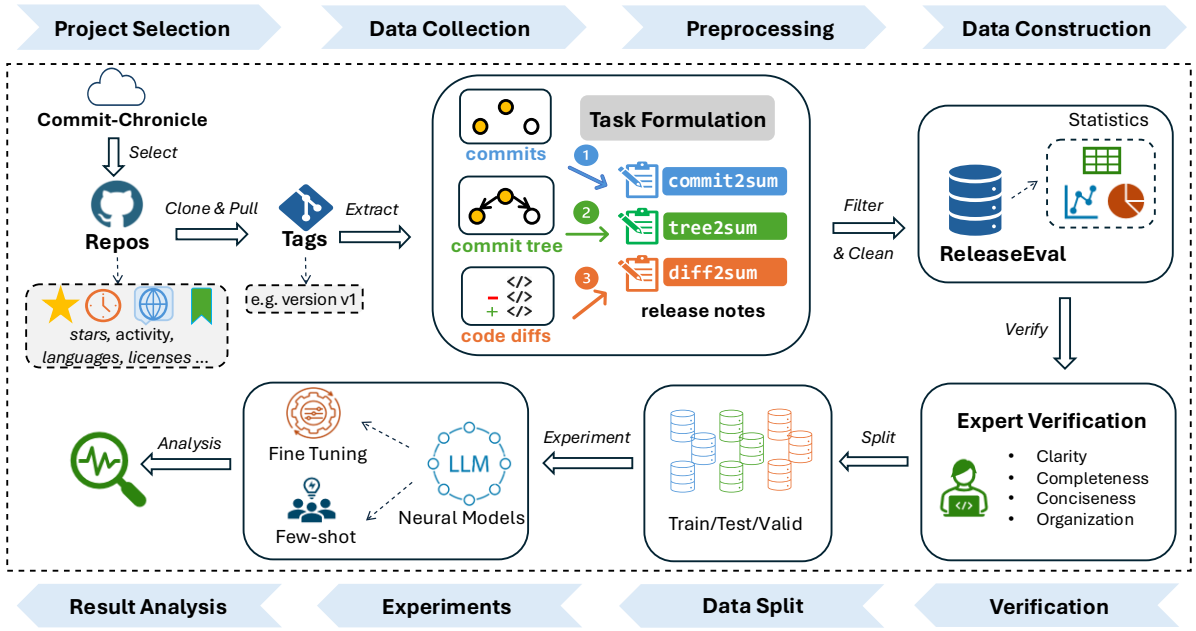


Figure 2: The overall framework of RELEASEEVAL: (1) selecting projects from the CommitChronicle dataset (Eliseeva et al., 2023); (2) collecting raw data of releases, commits, and code diffs based on the selected projects; (3) preprocessing the data using filtering strategies; (4) generating dataset statistics; (5) verifying dataset quality through expert evaluation; (6) splitting the dataset for training and evaluation; (7) conducting experiments; and (8) analyzing and discussing the results.

3.2.3 Dataset Processing

To obtain high-quality data, we apply several filtering strategies targeting the following attributes.

Versions. The first step is to extract and clean repository versions, enabling mapping of release notes to corresponding commits. For example, the release notes for version 1.1.1 are based on the commits between the previous version (e.g., 1.1.0) and the current version (1.1.1). Cleaning versions helps filter out pre-release or test versions (e.g., "beta," "alpha," or "rc"). Similar to filtering strategy used by (Kamezawa et al., 2022), we use regular expressions to identify and remove invalid versions, followed by validating core version numbers using the packaging.version library. After filtering, the valid versions are sorted chronologically, and only consecutive version pairs are retained.

Release Notes. Based on the cleaned versions, we retrieve the corresponding release notes using the GitHub API. Based on previous work (Nath and Roy, 2022; Kamezawa et al., 2022; Daneshyan et al., 2025), we choose the following three subsequent filtering strategies to clean the release notes. (1) *Coarse Filtering:* We first apply general filtering criteria to remove invalid release notes, including those that are empty, consist of only one sentence, are duplicates, or contain fewer than 10 tokens. We also discard overly long release notes to save computational resources (Kamezawa et al., 2022). These cases represent less than 3% of the dataset and have negligible effect on data coverage and representativeness. (2) *Category-based Filtering:* To ensure semantic coverage and informativeness, we adopt the category-matching tool proposed by Kamezawa et al. (2022) to exclude release notes that cover fewer than two of the following categories: features, improvements, bug fixes, and deprecations. (3) *Noise Filtering:* For the remaining release notes, we remove token-level noise such as issue versions, emojis, and SHA identifiers, following prior commit filtering studies (Jiang et al., 2017; Xu et al., 2019; Wang et al., 2021; Shi et al., 2022; Dong et al., 2022).

Commits. Using the retained consecutive version pairs, we employ PyDriller to collect commit messages, including the corresponding commit hash, author, timestamp, and code diffs. We apply the same *noise filtering* strategy to commit messages as to release notes.

Commit Trees. Similarly, using consecutive version pairs, we access commit tree information through the GitHub API, applying the same fil-

tering strategy as we use for commit messages to remove irrelevant information. The commit tree is stored and presented in a plain text format, using the symbols specified in Table 7 (see in Appendix A) to represent the structure.

Code Diffs. We construct diffs at the version level by combining file-level modifications with the corresponding code changes, where each modification records the file path, change type (e.g., modify, add and delete), and related code changes.

3.2.4 Dataset statistics

After processing, Table 2 shows that the dataset includes 3,369 repositories, resulting in 94,987 release notes and commit trees, alongside 1,074,577 commit messages and code diffs. Based on the average length of key attributes, the code diffs are significantly longer than other input types, with an average length exceeding 1,300 tokens. Figure 3 illustrates the distribution of programming languages in RELEASEEVAL, with TypeScript and JavaScript being the most dominant.

3.2.5 Quality verification

Finally, we conduct an expert verification to ensure the data quality. Following Daneshyan et al. (Daneshyan et al., 2025), we used four metrics rated on a 5-point Likert scale for evaluation. (1) *Completeness*: Ensures the release note includes every significant change made between versions. (2) *Clarity*: Verifies that each update is clearly and correctly presented. (3) *Conciseness*: Confirms the note delivers only key changes without unnecessary details. (4) *Organisation*: Evaluates whether the content is arranged in a clear and logical structure. We invited three experts (each with 6–8 years of programming experience) to independently and blindly assess 1,000 release notes randomly selected from 96 projects. To assess inter-rater reliability, we measured overall inter-annotator agreement, which reached 0.85 (Fleiss' Kappa (Fleiss, 1971)).

Expert review confirms the high quality of the benchmark, with evaluation results included in the released repository. Among all metrics, organisation and conciseness received the highest scores (4.74 and 4.33, respectively). Notably, over 95% of release notes in the dataset follow a standard structure with clearly defined categories. Clarity achieved a moderate score (4.08), while completeness scored the lowest (3.85), which aligns with prior findings (Wu et al., 2022) that missing or incorrect change information is a common issue in release note practices. The relatively lower clarity scores observed in 68% of small-scale projects (e.g., storybook, vaadin, etc.) may be due to their direct reuse of raw commit messages without paraphrasing. In contrast, large-scale projects (e.g., Google, Kubernetes, etc.) tend to produce high-quality release notes across all four evaluation metrics. Despite variations in writing style, these notes generally follow a standard structure and present information clearly and coherently.

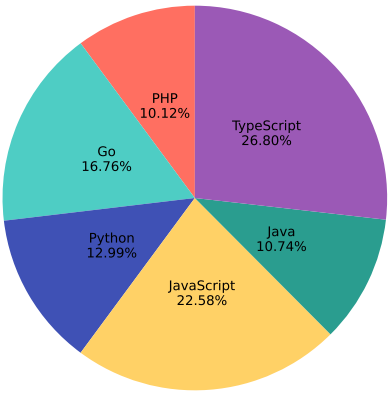


Figure 3: Programming Language Distribution of RELEASEEVAL.

Feature	Total
Repositories	3,369
Release Notes	94,987
Commit Trees	94,987
Commit Messages	1,074,577
Code diffs	1,074,577
Avg. # Commits per Release Note	14
Avg. # Tokens per Release Notes	198
Avg. # Tokens per Commit Message	14
Avg. # Tokens per Commit Tree	144
Avg. # Tokens per Code Difference	1,348

Table 2: Dataset Statistics of RELEASEEVAL

3.3 Task Formulation

We organize release note generation into a three-level task suite according to the granularity of software-change evidence. *commit2sum* follows the dominant setting in prior work: given the commit messages between two consecutive releases, the model generates the corresponding release note. This setting evaluates how models aggregate human-written descriptions of software changes.

tree2sum adds commit tree structure to the commit-message input, including branch divergence, merge points, and parent–child relations.

This setting evaluates whether models can exploit lightweight structural context beyond a flat sequence of commit messages.

diff2sum represents the most fine-grained level, where the input consists of raw code diffs between two releases. This setting evaluates whether models can abstract release-level information from code-level changes, which are more faithful to the implementation but also longer and noisier than commit messages. Together, the three tasks cover semantic, structural, and code-level evidence, enabling RELEASEEVAL to assess how models behave across different levels of software-change abstraction.

3.4 Data Split

We split the dataset into training, validation, and test sets using an 8:1:1 ratio, applied consistently across all programming languages. This yields 75,993 data for training, 9,495 for validation, and 9,499 for testing.

4 Experiments

4.1 Model Selection

To comprehensively assess the effectiveness of representative language models on RELEASEEVAL, we evaluate seven models, covering both encoder–decoder and decoder-only architectures. Specifically, we select BART (Lewis et al., 2020), a strong baseline previously applied to release note generation (Kamezawa et al., 2022), and T5 (Raffel et al., 2020), a widely used encoder–decoder model for summarization and other text generation tasks. For decoder-only LLMs, we include a set of open-source models with different scales and architectures: LLaMA3.1 (8B) and LLaMA3.3 (70B) (Touvron et al., 2024), Mistral (8B and 22B) (Jiang et al., 2023), and Qwen2.5 (7B, 32B, and 72B) (Qian et al., 2024).

4.2 Evaluation Metrics

To automatically evaluate the quality of the generated release notes, we use the widely adopted metrics BLEU-4 (Papineni et al., 2002), ROUGE-L (Lin, 2004), and METEOR (Banerjee and Lavie, 2005), which are commonly used in prior work (Nath and Roy, 2022; Kamezawa et al., 2022; Jiang et al., 2021) to capture lexical overlap and semantic similarity. For manual evaluation, we assess the generated release notes using four dimensions: Completeness, Clarity, Conciseness, and Organisation, as previously introduced in Section 3.2.5.

4.3 Implementation Details

We evaluate the models using both fine-tuning (Howard and Ruder, 2018) and few-shot settings (Brown et al., 2020). Specifically, due to limited computational resources, we fine-tune three smaller LLMs in the 7B–8B range (Qwen2.5-7B, LLaMA3.1-8B, and Mistral-8B), along with BART and T5. For few-shot evaluation, we apply zero-shot, two-shot, and four-shot prompting to all LLMs. In the zero-shot setting, all seven models are evaluated across all three tasks. For the two-shot and four-shot settings, we focus on the *commit2sum* and *tree2sum* tasks, excluding *diff2sum* due to its excessive input length in the few-shot setting. Table 10 summarizes the hyperparameter settings and hardware configurations, and Table 11 provides the prompt templates for the few-shot setting; both tables are included in Appendix B.

5 Results and Analysis

5.1 Performance of Fine-tuned LMs

Fine-Tuning: Table 3 presents the fine-tuning results on the three task settings. Overall, LLM-based models achieve higher scores than BART and T5 across all tasks, suggesting that larger pre-trained models are more effective for release note generation under different evidence granularities. Among the LLMs, LLaMA3.1-8B and Ministral-8B obtain comparable top-tier performance, with relatively small differences across several metrics. Therefore, we avoid over-interpreting the ranking among the strongest models and focus instead on the broader trend that supervised LLMs can learn release note generation effectively across semantic, structural, and code-level inputs.

We further fine-tune three code-specific 7B models as a supplementary analysis, with results reported in Appendix D. These models underperform the general-purpose LLMs across the three tasks, suggesting that release note generation requires not only code-level representation, but also multi-change aggregation and documentation-style abstraction.

5.2 Performance of Few-shot LLMs

Zero-shot: Table 4 presents the zero-shot performance of LLMs across three tasks. LLaMA3.1-8B and Mistral-8B achieve the highest scores in this setting, particularly on *tree2sum* (7.54% BLEU-4, 33.18% ROUGE-L, 29.49% METEOR). However, performance in the zero-shot setting remains

Method	commit2sum			tree2sum			diff2sum		
	BLEU-4	ROUGE-L	METEOR	BLEU-4	ROUGE-L	METEOR	BLEU-4	ROUGE-L	METEOR
BART	19.96	32.48	32.23	25.36	37.23	37.65	17.52	28.11	24.38
T5	22.43	34.02	34.13	27.61	37.75	38.83	18.20	26.37	24.29
Qwen-2.5-7B	36.42	48.39	50.51	41.19	51.47	54.05	19.00	28.11	36.94
LLaMA3.1-8B ★	**37.79**	**49.90**	**52.11**	42.64	52.91	55.37	**33.91**	**44.81**	**45.37**
Ministral-8B ❖	37.42	49.74	51.92	**42.77**	**53.14**	**55.79**	32.79	43.57	44.82

Table 3: Fine-tuning performance (%) of various language models on three tasks. Best performance is marked bold.

Model	commit2sum			tree2sum			diff2sum		
	BLEU-4	ROUGE-L	METEOR	BLEU-4	ROUGE-L	METEOR	BLEU-4	ROUGE-L	METEOR
7–8B Models									
Qwen2.5-7B	3.22	25.67	23.83	4.70	26.31	26.03	3.05	21.78	20.48
LLaMA3.1-8B ★	**6.07**	29.85	**27.23**	**7.54**	29.73	**29.49**	5.12	25.78	21.04
Ministral-8B ❖	5.39	**31.43**	25.94	7.05	**33.18**	27.48	**5.29**	**30.02**	**23.79**
22–32B Models									
Mistral-22B	4.46	28.35	25.79	5.70	30.17	26.09	4.37	26.71	24.83
Qwen2.5-32B	3.14	26.54	24.29	4.69	26.88	26.17	3.52	27.71	24.70
70–72B Models									
Llama3.3-70B	5.65	27.79	26.31	7.49	30.27	29.46	5.21	27.00	25.94
Qwen2.5-72B	3.43	27.40	24.31	3.35	26.80	23.19	3.38	26.32	23.12

Table 4: Zero-shot performance (%) of LLMs at different scales on three tasks. Best performance is marked bold.

significantly lower than that of fine-tuned models, especially in BLEU-4, indicating that models capture high-level semantics but lack lexical fidelity. **Few-shot:** Figure 4 presents few-shot performance on *commit2sum* and *tree2sum*. Compared with fine-tuning, few-shot prompting shows higher sensitivity to prompt construction and input length. Although additional examples improve performance for some models, the trend is not consistently monotonic, especially for smaller models. This suggests that few-shot release note generation is affected not only by model capability, but also by the interaction between exemplar length, context budget, and software-change input verbosity.

5.3 Impact of LLM Scale

As shown in Figure 4, smaller models such as LLaMA3.1-8B and Qwen2.5-7B exhibit noticeable fluctuations in the few-shot setting. In contrast, larger models (e.g., Qwen2.5-72B, LLaMA3.3-70B) tend to perform more consistently with shots increasing, likely due to their higher parameter capacity and greater robustness to longer inputs.

5.4 Impact of inputs on Release Note Quality

According to the results presented in Table 3, Table 4, and Figure 4, the three task settings expose different aspects of model behavior. *tree2sum* generally obtains strong performance because it preserves the semantic content of commit messages while adding lightweight structural information. This suggests that commit history structure is useful and relatively accessible to current LLMs. In contrast, *diff2sum* uses raw code diffs as its evidence source. Its lower automatic scores should not be interpreted simply as evidence that diffs are less informative; rather, they indicate that current models still struggle to select and abstract release-level information from long, low-level, and noisy code changes.

Case Study: We present a representative example from fine-tuned Ministral-8B in Table 12 (see Appendix C), which shows the generated release notes for all three tasks alongside the corresponding reference note. Among the three inputs, *tree2sum* achieves the highest BLEU-4 score (66.28), demonstrating strong alignment with the reference and capturing nearly all critical changes. *commit2sum* also identifies several key updates (BLEU-4: 39.95), though it introduces some unrelated information. *diff2sum*, by contrast, achieves the lowest score (BLEU-4: 16.10), missing several important updates and reflecting the difficulty of abstracting accurate summaries from raw code diffs. Despite differences in quality, all three outputs follow a consistent release note format, indicating that the model has effectively learned the standard

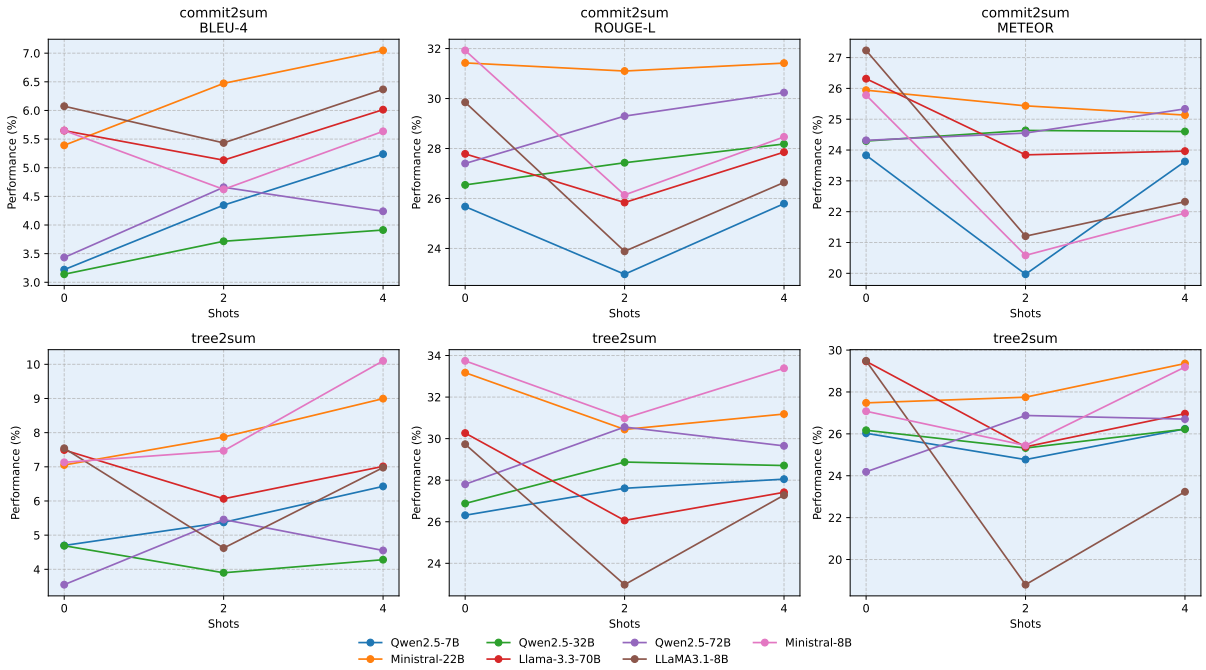


Figure 4: Few-shot performance (%) of LLMs at different scales on the *commit2sum* and *tree2sum* tasks across zero-, two-, and four-shot settings.

Data	Comp.	Clar.	Conc.	Org.
commit2sum	3.83	3.59	3.45	4.23
tree2sum	4.02	3.68	3.47	4.23
diff2sum	3.65	2.79	3.06	4.18

Table 5: Human evaluation results of release note quality across three tasks based on completeness (Comp.), clarity (Clar.), conciseness (Conc.) and organisation (Org.).

structure and writing style during fine-tuning. To complement these insights, we conducted a quantitative evaluation with domain experts to assess the effect of input types on generation quality.

Quantitative Evaluation: Following the evaluation protocol introduced in Section 3.2.5, three experts evaluated 100 randomly selected outputs from fine-tuned Ministral-8B using four metrics: completeness, clarity, conciseness, and organisation. The evaluation yielded Fleiss' kappa scores ranging from 0.82 to 0.89, indicating strong inter-rater agreement. The results are displayed in Table 5, among the three tasks, *tree2sum* achieves the highest overall quality, especially in completeness (4.02) and clarity (3.68), and performs comparably to *commit2sum* in terms of conciseness and organisation. These results are consistent with our earlier findings, confirming the advantage of structural input for release note generation. Interestingly, *diff2sum* achieves a relatively high score in completeness (3.65), indicating that code diffs provide rich information. However, outputs based on code diffs are frequently redundant and repetitive, resulting in lower scores for clarity (2.79) and conciseness (3.06). This highlights a trade-off between content coverage and readability when using fine-grained but long inputs like code diffs. These results also demonstrate that effectively identifying key changes from rich but lengthy code diffs remains a major challenge.

6 Conclusion

In this paper, we introduced RELEASEEVAL, a reproducible and license-aware multi-granularity benchmark for release note generation. By organizing release note generation into *commit2sum*, *tree2sum*, and *diff2sum*, RELEASEEVAL enables models to be evaluated under richer forms of software-change evidence. Our results show that LLMs can leverage structured commit evidence, while long and low-level code diffs remain difficult to abstract into release-level summaries. Expert evaluation further supports these observations by assessing completeness, clarity, conciseness, and organisation. Overall, RELEASEEVAL provides a unified benchmark for studying how language models transform different granularities of software-change evidence into release-level documentation.

7 Limitations

One limitation of our dataset concerns the nature of the ground-truth release notes. Prior work has shown that a clear and consistent style is important for readability (Wu et al., 2022), yet most of our annotated summaries follow a highly technical style that emphasizes low-level implementation details. While this style is appropriate for developers, it lacks broader contextual information, which reduces readability for non-technical stakeholders and limits the benchmark's suitability for evaluating release notes aimed at diverse audiences.

Another limitation lies in the scope of the *tree2sum* and *diff2sum* tasks. The *diff2sum* setting often involves very long code diffs that can exceed model input length limits, making it difficult to preserve important details. Although techniques such as input truncation (Liu and Lapata, 2019) and hierarchical encoding (Rae et al., 2019) have been proposed, effectively applying them in this domain remains challenging. The *tree2sum* setting introduces additional complexity due to the hierarchical structure of commit trees and their dependencies, which may be better represented with graph-based methods such as Graph Neural Networks (GNNs) (Scarselli et al., 2008).

Finally, our model evaluation is designed to characterize representative model behavior on RELEASEEVAL, rather than to provide an exhaustive study of model scaling or adaptation strategies. We therefore fine-tune representative 7B–8B LLMs and encoder–decoder baselines, while evaluating larger LLMs under zero-shot and few-shot settings. Extending parameter-efficient fine-tuning to larger models is a complementary direction for future work.

References

Abdulkareem Alali, Huzefa Kagdi, and Jonathan I. Maletic. 2008. What's a typical commit? a characterization of open source software repositories. In *16th IEEE International Conference on Program Comprehension (ICPC)*, pages 182–191.

Satanjeev Banerjee and Alon Lavie. 2005. 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*, pages 65–72. Association for Computational Linguistics.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 1877–1901.

Farbod Daneshyan, Runzhi He, Jianyu Wu, and Minghui Zhou. 2025. Smartnote: An llm-powered, personalised release note generator that just works. *Proceedings of the ACM on Software Engineering*, 2(FSE):1663–1686.

Jinhao Dong, Yiling Lou, Qihao Zhu, Zeyu Sun, Zhilin Li, Wenjie Zhang, and Dan Hao. 2022. Fira: fine-grained graph-based code change representation for automated commit message generation. In *Proceedings of the 44th International Conference on Software Engineering*, pages 970–981.

Aleksandra Eliseeva, Yaroslav Sokolov, Egor Bogomolov, Yaroslav Golubev, Danny Dig, and Timofey Bryksin. 2023. From commit message generation to history-aware commit message completion. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering*, pages 723–735. IEEE.

Joseph L Fleiss. 1971. Measuring nominal scale agreement among many raters. *Psychological bulletin*, 76(5):378.

Inc. GitHub. 2025. Github rest api v3 documentation. https://docs.github.com/en/rest. Accessed: 2025-05-31.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 328–339.

Yan Hu, Jun Zhang, Xiaomei Bai, Shuo Yu, and Zhuo Yang. 2016. Influence analysis of github repositories. *SpringerPlus*, 5(1):1268.

Huaxi Jiang, Jie Zhu, Li Yang, Geng Liang, and Chun Zuo. 2021. Deeprelease: Language-agnostic release notes generation from pull requests of open-source software. In *2021 28th Asia-Pacific Software Engineering Conference*, pages 101–110. IEEE.

Siyuan Jiang, Ameer Armaly, and Collin McMillan. 2017. Automatically generating commit messages from diffs using neural machine translation. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 135–146. IEEE.

Yinhan Jiang, Naman Goyal, Thomas Scialom, Andrea Muti, Yichong Qian, Daniel Simig, Albert Webson, and 1 others. 2023. Mistral 7b. https://mistral.ai/news/announcing-mistral-7b/.

Hisashi Kamezawa, Noriki Nishida, Nobuyuki Shimizu, Takashi Miyazaki, and Hideki Nakayama. 2022. Rnsum: A large-scale dataset for automatic release note generation via commit logs summarization. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8718–8735.

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*, pages 19–22.

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.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out: Proceedings of the ACL-04 workshop*, pages 74–81. Association for Computational Linguistics.

Yang Liu and Mirella Lapata. 2019. Text summarization with pretrained encoders. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3721–3731.

Rada Mihalcea and Paul Tarau. 2004. Textrank: Bringing order into text. In *Proceedings of the 2004 conference on empirical methods in natural language processing*, pages 404–411.

N Moratanch and S Chitrakala. 2017. A survey on extractive text summarization. In *2017 international conference on computer, communication and signal processing (ICCCSP)*, pages 1–6. IEEE.

Laura Moreno, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Andrian Marcus, and Gerardo Canfora. 2016. Arena: an approach for the automated generation of release notes. *IEEE Transactions on Software Engineering*, 43(2):106–127.

Sristy Sumana Nath and Banani Roy. 2022. Towards automatically generating release notes using extractive summarization technique. *arXiv preprint arXiv:2204.05345*.

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 on Association for Computational Linguistics*, pages 311–318. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Yujia Qian, Yichong Xie, Xu Han, Yitao Zhu, Shaohan Liu, Weizhu Chen, and 1 others. 2024. Qwen2: The next generation of qwen models. *Preprint*, arXiv:2405.10050.

Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. 2019. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations (ICLR)*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21:1–67.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80.

Ensheng Shi, Yanlin Wang, Wei Tao, Lun Du, Hongyu Zhang, Shi Han, Dongmei Zhang, and Hongbin Sun. 2022. Race: Retrieval-augmented commit message generation. *arXiv preprint arXiv:2203.02700*.

Davide Spadini, Maurício Aniche, and Alberto Bacchelli. 2018. Pydriller: Python framework for mining software repositories. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 908–911.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Junjie Chan, Jie Fu, and 1 others. 2024. Llama 3: Open foundation and instruction-tuned language models. *Preprint*, arXiv:2404.14219.

Haoye Wang, Xin Xia, David Lo, Qiang He, Xinyu Wang, and John Grundy. 2021. Context-aware retrieval-based deep commit message generation. *ACM Transactions on Software Engineering and Methodology*, 30(4):1–30.

Jianyu Wu, Hao He, Wenxin Xiao, Kai Gao, and Minghui Zhou. 2022. Demystifying software release note issues on github. In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*, pages 602–613.

Shengbin Xu, Yuan Yao, Feng Xu, Tianxiao Gu, Hanghang Tong, and Jian Lu. 2019. Commit message generation for source code changes. In *IJCAI*.

Liguo Yu. 2009. Mining change logs and release notes to understand software maintenance and evolution. *CLEI Electronic Journal*, 12(2):1–10.

Attribute	Description
	Release information
repo	The URL of the repository.
version_url	The URL linking to the specific release version in the repository.
release_note	The content of the release note.
	Commits information
commit_url	The URL linking to the comparison of commits between versions.
hash	The unique identifier (hash) of the commit.
message	A brief description of the changes introduced by the commit.
author	The person who authored the commit.
time	The date and time of the commit.
commit_tree	The full commit tree between two versions (branches and merges).
	Code diff information
mod	File modification type (added, modified, deleted).
file	Target modified files.
diff	The actual code changes introduced by the commit.

Table 6: Basic information for each entry in RELEASEEVAL, including release information, commit information, and code diff information.

Symbol	Description
*	Represents a unique commit.
| \	Shows branch divergence (split or merge).
| /	Marks a point where branches merge back.
| *	Indicates a commit on a branch before it merges back.

Table 7: Symbols in commit tree

A Dataset Details

Table 6 summarizes all dataset attributes and their descriptions. Compared to existing datasets, RELEASEEVAL provides more comprehensive information by including not only release notes and commit messages, but also commit metadata, commit trees, and fine-grained code diffs. Table 7 illustrates the common symbols used in commit trees, including unique commits, branch divergence, and merge points.

B Experiments Details

Table 10 summarizes the hyperparameters for both fine-tuning and few-shot settings, while Table 11 details the few-shot prompts.

C Results Details

Table 12 presents a case study comparing release notes from a fine-tuned Ministral-8B model on the three tasks (*commit2sum*, *tree2sum*, and *diff2sum*). Colored highlights indicate ground-truth overlap, illustrating how input types affect coverage and quality.

D Additional Experiments

To provide a comprehensive evaluation of model behavior on RELEASEEVAL, we conduct two supplementary experiments following settings in Table 10. First, we evaluate chain-of-thought (CoT) zero-shot prompting using the same decoding setup to examine whether explicit reasoning helps release note generation. Second, we fine-tune code-specific language models using the same LoRA training setup and three-run averaging to test whether code-oriented pretraining is better suited for this task. These experiments complement the main results by analyzing reasoning behavior and model specialization under a consistent evaluation protocol.

CoT zero-shot prompting. We evaluate 7–8B LLMs with chain-of-thought (CoT) zero-shot prompting on all three tasks, using two prompt variants to check robustness. Results are reported in Table 8. Overall, CoT prompting shows task-dependent effects rather than uniform improvements. On *commit2sum*, CoT brings only small changes compared with standard zero-shot prompting, likely because commit messages already provide compact, human-written semantic descriptions. On *tree2sum*, CoT is more helpful, especially for LLaMA3.1-8B, suggesting that explicit reasoning can help models follow hierarchical commit-tree structure and organize changes across branches, merges, and parent–child relations. However, CoT does not consistently improve *diff2sum*. For long and low-level code diffs, chain-style reasoning may introduce verbosity or

Model	commit2sum			tree2sum			diff2sum		
	BLEU-4	ROUGE-L	METEOR	BLEU-4	ROUGE-L	METEOR	BLEU-4	ROUGE-L	METEOR
Qwen2.5-7B	3.22	25.70	23.70	4.69	26.60	26.15	3.01	21.80	21.33
LLaMA3.1-8B	**6.65**	**32.10**	**28.90**	**7.91**	**30.85**	**29.49**	**5.50**	26.10	22.50
Ministral-8B	5.11	29.90	24.70	6.35	28.95	25.10	3.40	**27.20**	**23.00**

Table 8: CoT zero-shot performance (%) of 7–8B LLMs on the three ReleaseEval tasks.

Model	commit2sum			tree2sum			diff2sum		
	BLEU-4	ROUGE-L	METEOR	BLEU-4	ROUGE-L	METEOR	BLEU-4	ROUGE-L	METEOR
Qwen3-Coder	29.83	42.93	45.10	34.85	46.01	47.88	17.91	27.85	31.10
DeepSeek-Coder	**31.25**	**44.80**	**45.94**	**36.38**	**46.62**	**49.11**	**19.40**	**30.93**	**35.27**
CodeGemma	31.09	43.28	45.57	35.55	46.10	47.75	18.02	29.50	33.70

Table 9: Fine-tuning performance (%) of 7B code-specific language models on the three ReleaseEval tasks. Results are averaged over three runs.

Category	**Configuration**
	Fine-Tuning
Adaptation Method	LoRA
Learning Rate	1×10^{-4}
Batch Size	16
Early Stopping	Monitor validation loss (patience = 2)
Hardware	4× NVIDIA A100 GPUs
Repetitions	3 runs (results averaged)
	Few-Shot
Max Generation Length	256 tokens
Temperature	0.0
Top-p	1.0
Prompt Variants	Zero-shot, two-shot, four-shot

Table 10: Configuration of fine-tuning and few-shot settings.

Type	**Prompt**
Zero-shot	Based on the given <X>, generate a release note summarizing the key changes, including bug fixes, new features, enhancements, and other modifications.
Few-shot	Based on the given <X>, generate a release note summarizing the key changes, including bug fixes, new features, enhancements, and other modifications. I will provide you with <N> example(s): [*Exemplars*]

Table 11: Prompts for LLMs, where <X> is the commit message, commit tree, or code difference, and <N> is the total number of examples.

amplify irrelevant implementation details, distracting the model from identifying release-note-worthy changes. These results indicate that reasoning prompts are more useful for structured evidence than for raw code-diff inputs.

Code-specific language models. We further fine-tune three 7B code-specific models, Qwen3-Coder, DeepSeek-Coder, and CodeGemma, on all three tasks, with results averaged over three runs in Table 9. These models perform below the strongest general-purpose LLMs in Table 3, including on *diff2sum*. This suggests that release note generation is not simply a code understanding task. Although code-specific models are pretrained to model source code and capture syntax-level or local semantic regularities, release note generation requires broader abilities: aggregating multiple commits or diffs, selecting release-relevant changes, abstracting low-level implementation details, and producing concise documentation-style summaries. The weaker performance on *diff2sum* is especially informative, since this setting directly uses raw code diffs. It indicates that code-oriented pretraining alone is insufficient for transforming detailed code edits into release-level descriptions.

Task	Release Notes	BLEU-4
Ground Truth	What's Changed fix: prevent all referenced properties from updating to latest regardless of recipe by @ajohnsonz Add headers for FindCommitters data tables by @pstreef feat: add from date option to FindCommitters recipe by @pstreef Handle empty string option by @pstreef Make DependencyUseStringNotation compatible with G.GString by @Jenson3210 Normalization of maven repositories that don't permit directory listing by @jkschneider New Contributors @Jenson3210 made their first contribution Full Changelog: v8.14.3...v8.14.4	
commit2sum	What's Changed Fix: prevent all referenced properties from updating to latest regardless of recipe by @timtebeek Add headers for FindCommitters data tables by @eduardoramirez Fix UpgradeDependencyVersion so it doesn't use the same GradleProject marker for every project in a multi- project build. by @timtebeek Add logic to simplify J.Ternary by @jkschneider feat: add from date option to FindCommitters recipe by @eduardoramirez Handle empty string option by @eduardoramirez Add recipe which encloses all parameters interpolated into Groovy GStrings in curly braces by @timtebeek Add FindClassHierarchy recipe to produce a data table of class declarations, their superclasses, and interfaces by @timtebeek Make DependencyUseStringNotation compatible with G.GString by @JenteSondervorst Improve AddProperty to insert in alphabetical order by key by @timtebeek Normalization of maven repositories that don't permit directory listing by @sambsnyd New Contributors @JenteSondervorst made their first contribution Full Changelog: v8.23.2...v8.23.3	39.95
tree2sum	What's Changed fix: prevent all referenced properties from updating to latest regardless of recipe by @timtebeek Add headers for FindCommitters data tables by @knutwannheden feat: add from date option to FindCommitters recipe by @josh-farrant Handle empty string option by @knutwannheden Make DependencyUseStringNotation compatible with G.GString by @timtebeek Improve AddProperty to insert in alphabetical order by key by @timtebeek Normalization of maven repositories that don't permit directory listing by @knutwannheden New Contributors @josh-farrant made their first contribution Full Changelog: v8.7.1...v8.7.2	**66.28**
diff2sum	What's Changed Add from date option to findCommitters by @timtebeek Add support for gradle plugin version notation by @timtebeek Fix SimplifyBooleanExpressionVisitor by @timtebeek New Contributors ... @tobiasnolte made their first contribution Full Changelog: v8.3.2...v8.3.3	16.10

Table 12: Example release notes generated by a fine-tuned Ministral-8B on ReleaseEval across three tasks. Colors indicate overlap with the ground truth (blue),*commit2sum* (orange), *tree2sum* (green), and *diff2sum* (pink).