SmartNote: An LLM-Powered, Personalised Release Note Generator That Just Works

FARBOD DANESHYAN*, Peking University, China
RUNZHI HE*, Peking University, China
JIANYU WU, Peking University, China
MINGHUI ZHOU†, Peking University, China

The release note is a crucial document outlining changes in new software versions. It plays a key role in helping stakeholders recognise important changes and understand the implications behind them. Despite this fact, many developers view the process of writing software release notes as a tedious and dreadful task. Consequently, numerous tools (e.g., DeepRelease and Conventional Changelog) have been developed by researchers and practitioners to automate the generation of software release notes. However, these tools fail to consider project domain and target audience for personalisation, limiting their relevance and conciseness. Additionally, they suffer from limited applicability, often necessitating significant workflow adjustments and adoption efforts, hindering practical use and stressing developers. Despite recent advancements in natural language processing and the proven capabilities of large language models (LLMs) in various code and text-related tasks, there are no existing studies investigating the integration and utilisation of LLMs in automated release note generation. Therefore, we propose SmartNote, a novel and widely applicable release note generation approach that produces high-quality, contextually personalised release notes by leveraging LLM capabilities to aggregate, describe, and summarise changes based on code, commit, and pull request details. It categorises and scores (for significance) commits to generate structured and concise release notes of prioritised changes. We conduct human and automatic evaluations that reveal SmartNote outperforms or achieves comparable performance to DeepRelease (state-of-the-art), Conventional Changelog (off-the-shelf), and the projects' original release note across four quality metrics: completeness, clarity, conciseness, and organisation. In both evaluations, SmartNote ranked first for completeness and organisation, while clarity ranked first in the human evaluation. Furthermore, our controlled study reveals the significance of contextual awareness, while our applicability analysis confirms SmartNote's effectiveness across diverse projects.

CCS Concepts: • **Software and its engineering → Documentation**; *Software evolution*; • **Computing methodologies → Information extraction**; **Natural language generation**.

Additional Key Words and Phrases: Release Notes, Automatic Software Engineering, Large Language Model

ACM Reference Format:
Farbod Daneshyan, Runzhi He, Jianyu Wu, and Minghui Zhou. 2025. SmartNote: An LLM-Powered, Personalised Release Note Generator That Just Works. *Proc. ACM Softw. Eng.* 2, FSE, Article FSE075 (July 2025), 24 pages. https://doi.org/10.1145/3729345

*Both authors contributed equally to this paper.
†Corresponding author.

Authors' Contact Information: Farbod Daneshyan, Peking University, Beijing, China, fabs@stu.pku.edu.cn; Runzhi He, Peking University, Beijing, China, rzhe@pku.edu.cn; Jianyu Wu, Peking University, Beijing, China, jianyu.wu@pku.edu.cn; Minghui Zhou, Peking University, Beijing, China, zhmh@pku.edu.cn.

ACM 2994-970X/2025/7-ARTFSE075
https://doi.org/10.1145/3729345

1 Introduction

Accompanying each new software version is a crucial document, the release note. Release notes summarise the new features, enhancements, bug fixes, and other changes in a software update, conveying the rationale and impact of changes to downstream developers and end users [Moreno et al. 2017; Wu et al. 2023], effectively serving as a software trail [Abebe et al. 2016]. To make informed update decisions, stakeholders evaluate the benefits of the update, such as bug fixes, features, and performance advancements, against potential drawbacks, like breaking changes that hinder the adoption of the new release [Bernard 2012]. Additionally, project managers use release notes to track the development progress and set milestones for release targets [Bi et al. 2020].

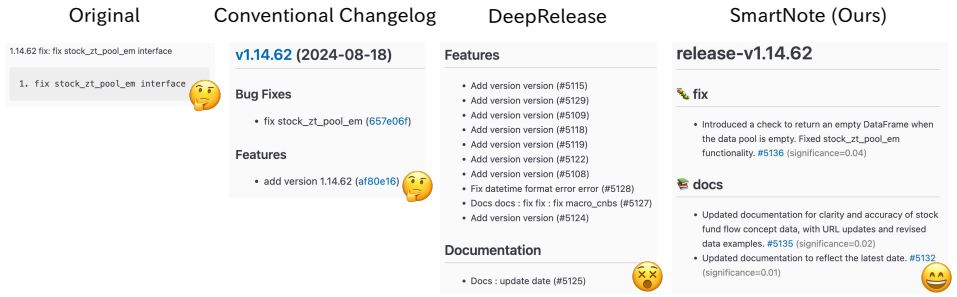


Fig. 1. Release Notes Generated for Version 1.14.62 of the AKShare Project [2024].

However, the task of writing release notes requires careful deliberation. Many developers and maintainers are reluctant as they perceive writing release notes as time-consuming and tedious; therefore, release notes are often neglected [Khomh et al. 2015]. Developers have expressed their frustrations, with one Microsoft developer stating in a blog post [Greenaway 2018], *"We hate creating release notes when it's our turn to ship software updates. This can be quite a challenge and very time-consuming"*. Empirical evidence supports this claim. Moreno et al. [2014] found that it takes up to 8 hours for an experienced developer to draft a release note document. This challenge is even more pronounced for the maintainers of large-scale open-source projects. For example, maintainers of *espressif/esp-idf* are required to gather information from more than 5,000 commits to compile a single release note document for *v5.2*. With the rise of continuous software delivery — a software engineering approach that emphasises rapid, efficient, and reliable service improvements [Chen 2015; Humble and Farley 2010] — frequent software releases are becoming the new norm. This shift places an increasing burden on developers to consistently produce high-quality release notes. In general, release note producers face the time-consuming and challenging task of creating a well-balanced, high-quality release note, while ensuring that it is suitable for their target audience [Wu et al. 2022].

Automated release note generation tools such as **DeepRelease** [Jiang et al. 2021] and **Conventional Changelog** [2024], have been proposed and implemented by researchers and practitioners. However, they lack widespread adoption. Sumana et al. [2024], interviewed respondents who indicated that *automated tools are not really useful to reduce work stress*. Similarly, Bi et al. [2020] found that none of the participants they interviewed had adopted such tools for automatically generating release notes. The lack of adoption may be due to limitations that hinder their real-world application. **First**, they often fail to consider the diverse needs of various audiences and project domains, which are important contextual factors for effective **personalisation**. For example, release note users and producers have different expectations on releases of different types (i.e., major, minor, patch) from projects of different domains (e.g., applications, tools) [Bi et al. 2020; Klepper et al. 2016; Wu et al.

2022]. This mismatch results in release notes lacking critical details for specific user groups and leaves users frequently overwhelmed with generic and irrelevant information. For instance, users of Libraries & Frameworks prefer more prevalent changes documented in release notes, while users of Software Tools prioritise information about performance and security improvements [Nath and Roy 2022; Wu et al. 2023]. **Second**, existing tools have limited **applicability**, demand extensive adoption efforts, require significant workflow adjustments, and often produce overly verbose outputs. For instance, ARENA only supports specific programming languages [Moreno et al. 2017], and DeepRelease excludes 46% of projects that do not adhere to pull request (PR) practices [Jiang et al. 2021]. Our results confirm that DeepRelease failed for ~10% of the projects we analysed. Other tools rely on basic automation that aggregates changes directly from version control systems, such as Git commit messages. However, this approach fails to adequately communicate the impact of changes [OpenStack 2022] and generates overly verbose release notes which lead to disengagement and difficulties in pinpointing significant and relevant changes [Nath and Roy 2024; Wu et al. 2023]. Moreover, a common frustration with many off-the-shelf release note generators, like Conventional Changelog [2024] and semantic-release [2024], is that they enforce commit message conventions, such as conventional commits [2024] or angular [2020] respectively, and require rigorous configuration. These approaches restrict usability by imposing requirements and pressuring developers to modify their workflows or design independent solutions. For instance, our evaluation reveals that Conventional Changelog fails for more than half the projects analysed. **Third**, current approaches do not explore the **effectiveness of LLMs** in release note generation. To the best of our knowledge, despite recent advancements in natural language processing (NLP) and the proven capabilities of LLMs in various code and text-related tasks [Achiam et al. 2024; Anthropic 2024; Touvron et al. 2023; Zhao et al. 2023], there are no existing studies investigating integration and utilisation of LLMs in automated release note generation. LLMs have significant benefits in code comprehension tasks that allow them to capture semantic meaning and connections between code and natural language text [Feng et al. 2020; Nam et al. 2024; Wang et al. 2021], allowing for intricate summaries of changes. Therefore, the use of LLMs could offer benefits in aggregating information from various sources and enabling rich, high-quality, personalised release notes by automating the interpretation of complex commit histories, extracting the most relevant information, and incorporating writing styles and organisational patterns to automatically generate concise and accurate release notes.

Thus, to bridge the gap in existing literature and support open source software (OSS) maintainers, we propose SmartNote, a release note generator that utilises the highly effective code to natural language comprehension capabilities of LLMs [Nam et al. 2024] to produce personalised, high-quality release notes for diverse GitHub projects. Notably, SmartNote infers optimal settings based on project context and captures the semantics of code, commit messages, and pull requests. This enables it to generate comprehensive and contextually relevant release notes, as illustrated in Figure 1, even for projects where commit messages are inconsistent, unstructured, or non-compliant with conventional standards, enhancing its **applicability**. Moreover, SmartNote is designed to accommodate the preferences of various project domains and release types [Wu et al. 2023], ensuring that generated release notes are contextually tailored in terms of content, organisation, and style. At a high level, SmartNote achieves this in four steps: 1) context comprehension, 2) change summarisation, 3) change categorisation, and 4) change filtration to remove less significant entries and details. These steps are carried out through a five-stage generation pipeline, as we will explain later in Section 3.2. For this study, we selected OpenAI's state-of-the-art LLM, "gpt-4o", which ranks #1 on the LMArena leaderboard [LMSYS and SkyLab 2024]. To optimise the prompts, we conducted multiple iterations of trial-and-error, guided by best practices in prompt engineering proposed in previous studies [Ekin 2023; Wei et al. 2024] and by LLM vendors [Sanders and Fishman 2023].

To evaluate SmartNote, we analyse generated release notes for completeness, clarity, conciseness, and organisation by conducting human and automated evaluations on 23 open source projects against baselines — DeepRelease, Conventional Changelog, and the projects' original release notes. To begin with, we find that SmartNote is applicable to all evaluated projects, whereas DeepRelease fails for ~10% and Conventional Changelog for ~54% of projects. Furthermore, the human evaluation indicates that over 80% of participants agree or strongly agree that SmartNote achieves the best results for completeness, clarity, and organisation, while conciseness ranks second. In the automated evaluation, we found that SmartNote achieves 81% commit coverage, ranking first in organisation with an information entropy of 1.59. In contrast, conciseness ranks third, yielding mixed results. This can be attributed to the compromises made by release note authors, who often improve conciseness by reducing commit coverage. Moreover, to assess the impact of context awareness on release note quality, we conducted an ablation study with both automatic and human evaluations, comparing SmartNote's generated release note with three variants. The results reveal that SmartNote captures human-written nuances, highlighting the importance of prompt engineering and context comprehension key components to SmartNote's effectiveness, particularly in completeness and clarity.

To summarise, despite industry and academic advances in automated release note generation, a gap remains between current methods and developer expectations. SmartNote resolves the current challenges by introducing the following innovations:

1. Workflow-Agnostic Design: Unlike prior tools that rely on rigid workflows (e.g., pull-merge strategies used by only 54% of OSS projects [Jiang et al. 2021] or commit conventions which are varied and not widely adopted), SmartNote works out-of-the-box with broader applicability (e.g., of the projects we evaluated, DeepRelease fails for more than 10% and Conventional-Changelog fails for more than 50%).

2. User-Centric: SmartNote generates clear, complete, and well-organised release notes, making them user-friendly, actionable, and tailored to diverse audiences, addressing gaps that hinder the adoption of state-of-the-art tools.

3. Tailored Pipeline: SmartNote uses a tailored pipeline to aggregate, classify, score commits for significance, merge related changes, and organise release notes based on project domain as defined by previous research [Wu et al. 2023] and context. This approach ensures personalised, concise, and structured release notes.

4. Prompt Engineering: Research shows that prompt engineering improves LLM performance [Liu et al. 2023; Schulhoff et al. 2024, 2023; Wei et al. 2024]. Without it, LLMs produce inconsistent, verbose, and sometimes nonsensical results, whereas with it, they are better personalised and more applicable (Section 4, EQ2).

2 Background

In this section, we focus on, 1) studies analysing release note practices, characteristics, and usage; 2) state-of-the-art tools for automating release note generation and an overview of off-the-shelf release note generation tools; and 3) a summary concluding the limitations of current approaches.

2.1 Release Note Characteristics and Contents

Initially, release notes served as data sources for broader studies on software maintenance and evolution [Alali et al. 2008; Maalej and Happel 2010; Shihab et al. 2013; Yu 2009]. It has only been in recent years that researchers have turned their attention toward empirical studies that specifically examine release note practices and the development of automated generation techniques.

Existing studies aimed at analysing release notes have greatly enhanced collective understanding of the characteristics and usages of release notes. Releases contain various aspects of information that must be organised into categories, to form well-structured release notes, which positively

impact software activities [Bi et al. 2020], such as software evolution and continuous software delivery. However, release notes can be overwhelming and tedious to write at times, resulting in missed information or errors [Wu et al. 2022]. Additionally, research [Nath and Roy 2022] finds that release note content contains four main types of artefacts: issues (29%), PRs (32%), commits (19%), and common vulnerability and exposure (CVE) issues (6%). It also highlights that users prefer addressed issues to be summarised or rephrased, and that users' preference for release note contents differs based on their background, for example, bug fixes are essential for developers and testers. Additionally, several studies find that conciseness is an important factor [Aghajani et al. 2020] and that most release notes list only between 6% and 26% of issues addressed [Abebe et al. 2016].

Expectations for release notes vary across software project domains (such as application software, system software, libraries and frameworks, and system tools) and release types (major, minor, patch) [Wu et al. 2022]. These factors influence the structure and content of release notes, which in turn affects their applicability [Nath and Roy 2024]. To this extent, Wu et al. [2023] conducted research to analyse and identify how release notes are organised, in what way they are written, and the content of release notes concerning project domain and release types. The authors analysed 612 release notes from 233 GitHub projects. They found that 64.54% of release notes organise changes into hierarchical structures, sorted by change type, affected module, change priority (the most common), or a combination of the three. Additionally, they identify three types of writing styles, "expository" (directly list the title or content of change-related commits/PRs/issues), "descriptive" (rephrase the content of change-related commits/PRs/issues to increase the readability and summarise the content of similar commits/PRs/issues), and "persuasive" (provide additional information to help developers understand the changes, such as the rationale behind the changes, the impact of the changes) which provide additional information and enhanced explanations specific to the project domain. Finally, they analyse the contents of release notes from different project domains, finding "System Software" and "Libraries & Frameworks" projects more likely to record breaking changes, while "Software Tools" projects emphasise enhancements.

2.2 Automatic Release Note Generation

Researchers have developed various approaches and tools to help achieve and enhance release note automation. Klepper et al. [2016] introduced a semi-automatic approach to generating audience-specific release notes that integrates with continuous delivery workflows. While this method offers flexibility and reduces the burden on release note producers, it assumes ideal conditions and lacks detailed implementation guidance, which may limit its practical applicability in diverse development environments. For example, it assumes that the data sources (e.g., issue trackers and version control systems) are well-structured and consistently used by the development team, which may not always be the case in real-world projects.

Moreno et al. [2017] introduced ARENA (Automatic RElease Notes generAtor), a tool to automatically generate release notes for Java projects. In comparison, Nath et al [2021] used extractive text summarisation technique, TextRank, to automate release notes production. Their approach functions without predetermined templates and is language-agnostic. Similarly, Jiang et al. [2021], proposed DeepRelease, a language-agnostic tool utilising deep learning techniques to generate release notes from the textual information contained in PRs. The tool concatenates the title, description, and commit messages of a preprocessed PR as the input, summarising them into a single change entry for the release note. They found that 54% of the 900 open-source projects they analysed generate release notes based on pull requests. However, while it is language-agnostic, it requires developers to follow specific pull request development practices, to automatically generate release notes. This requirement excludes the remaining 46% of projects that do not adhere to these practices, a significant amount. In contrast, Kamezawa et al. [2022] introduced a summarisation approach

leveraging transformer-based sequence-to-sequence networks. This approach generates labelled release notes by summarising commit messages, making it adaptable to various projects without specific constraints. However, it places a significant burden on developers to consistently write high-quality commit messages.

Developers have also created off-the-shelf solutions like *Conventional Changelog* [2024], which by default requires users to adhere to the conventional commit convention [Joslin 2024]. While less intricate than those proposed in research studies, off-the-shelf solutions are widely adopted (e.g., *Conventional Changelog* has more than one million downloads per week [NPM 2024]) due to their accessibility and ease of integration into development workflows. Additionally, they are customisable to cater to different projects' needs, however, this flexibility necessitates extensive configuration due to their reliance on templates and labels. Moreover, these tools typically generate changelogs that list all changes, including minor revisions, using the commit message title to describe the change, whereas release notes focus on summarising the most significant changes in a new release, ensuring the impact is understandable for the user regardless of context [OpenStack 2022]. Both serve the same fundamental purpose. For example, Release Drafter [2024], provides users with the flexibility to configure labels and headings for their commits and release notes. However, this flexibility requires developers follow a specific commit pattern. Other tools such as *semantic-release* [2024], *changelogen* [2024], and *github-changelog-generator* [2024] operate on similar principles, enforcing commit conventions or specific patterns to generate changelogs.

2.3 Knowledge Gap

Despite the strong interest in automation, the application of release note generators in the open source world remains limited. Bi et al. [2020] found that none of the participants they interviewed had adopted release note automation tools, and Sumana et al. [2024]'s participants complain about the stress of release note production even with the assistance of automation. A major contributing factor to this phenomenon is the limitation of current automation approaches. Existing tools lack key features, are prone to errors, and are often difficult to configure [Aghajani et al. 2020; Wu et al. 2022]. Additionally, existing automation techniques often generate standardised patterns, which can confuse users and lead to hesitancy in adoption due to concerns for relevance and accuracy [Nath and Roy 2024].

To summarise, existing research on release notes has highlighted the importance of structured, well-organised content that caters to both release note producers and users. However, current tools while effective in aggregating information and categorising release notes, have the following limitations:

(1) fail to address key discrepancies between release note producers and users;
(2) do not personalise the content, structure and style to meet the diverse needs of various software domains and release types;
(3) have stringent requirements, such as commit convention, templates, labels, or PR strategies;
(4) require extensive configuration;
(5) some work only for certain programming languages. e.g., ARENA.

These limitations often hinder widespread adoption, resulting in inconsistent quality and detail in generated release notes. In short, current generation methods lack personalisation and applicability.

3 Approach

To address the gaps and limitations in automatic release note generation, we introduce SmartNote, a novel approach that leverages the comprehension capabilities of LLMs [Nam et al. 2024] to aggregate the "what" and "why" of changes in code, commits and pull requests, generating high-quality, personalised release notes while remaining language and workflow agnostic. To personalise release notes to individual projects and release types, SmartNote incorporates best practices in

content, writing style, and organisation identified in previous research [Wu et al. 2023]. In this section, we first introduce the key component of SmartNote, the LLM module. We then provide an overview, followed by a detailed explanation of each stage of SmartNote.

3.1 The LLM Module

The LLM module is integral to the generation pipeline, encapsulating capabilities for the release note generation process. It is responsible for formatting input by generating prompts based on predefined templates and contextual information, as well as parsing the output. Additionally, it manages the model's context size limit by limiting the token count for each commit diff. In cases where the limit is exceeded, SmartNote provides warnings. However, the token limit is sufficiently large, making occurrences rare in practice, except in scenarios such as the initial commit or when entire folders are moved or deleted. In our evaluation, the model's context size limit was exceeded in only 4 of the 23 projects, affecting no more than two commits. To ensure future compatibility, SmartNote is designed to function with any LLM. For this study, we employed OpenAI's "gpt-4o" model, which was considered state-of-the-art at the time and ranked #1 on LMArena [LMSYS and SkyLab 2024]. The following subsections detail the prompt engineering techniques employed and the approach used to identify optimal hyperparameters.

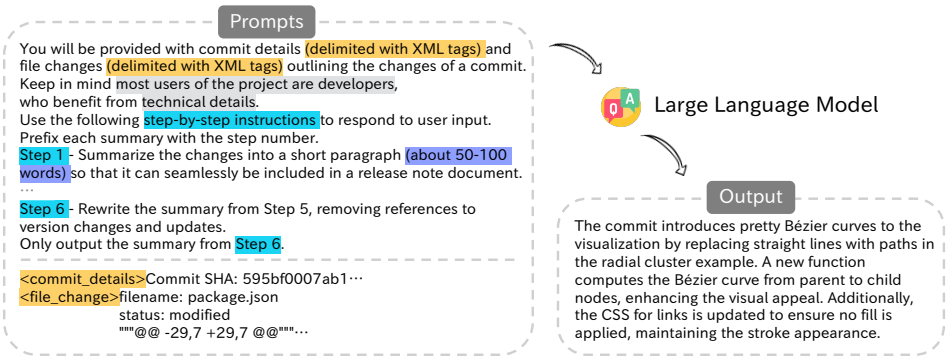


Fig. 2. Model Input and Output Example for Commit Summarisation based on the d3 Project [2024].

3.1.1 Prompt Engineering. To ensure high-quality responses from the LLM, we utilised prompt engineering techniques and best practices identified by research [Chen et al. 2024] and recommended by LLM vendors [OpenAI 2024b]. Additionally, we iterated using a trial-and-error process to identify the most suitable prompt template — manually analysing and evaluating the response, adjusting the prompt until the LLM outputs a correct and high-quality response. Due to the page limit, we are unable to elaborate on all of the prompt templates. However, we illustrate the prompt engineering tactics with the example of the commit summarisation task in Figure 2 (which has been shortened to highlight the key aspects of the prompt and accommodate page space constraints). Complete versions of the prompt templates are available in our replication package.

- **Delimiters**: Used to clearly indicate distinct parts of an input, especially for multi-line strings, helping to correctly distinguish the data that the model should focus on with the instructions given. The more complex a task is the more important it is to disambiguate task details. We use XML tags [OpenAI 2024b], to clearly indicate the code patch when generating release note entries and when summarising PRs to clearly indicate details such as the title and body, and highlight the commits.
- **Chain of Thought Prompting**: By providing a clear structure for the model to follow in the form of intermediate reasoning steps, the input helps guide the model's responses [Sahoo

et al. 2024; Wei et al. 2024]. To this extent, we split the complex task into simpler subtasks, and ask the model to follow a step by step guide to generate its response when summarising changesets.

- **One-Shot & Few-Shot Prompting**: Higher complexity tasks benefit from one-shot and few-shot prompting, a technique where one or a few examples are provided in the model input [Sahoo et al. 2024]. While examples don't always help [Liu et al. 2020; Reynolds and McDonell 2021], they do serve as a mechanism to further guide the model in recalling previously learnt tasks. We used few-shot prompting in the settings generator to determine the project domain [Brown 2020] and one-shot prompting for rephrasing entries and content.
- **Intent Classification**: Intent classification is a technique to identify the most relevant instruction for a query. In the case that the model needs to handle different cases, it is beneficial to categorise those cases ahead of time by hard coding it in the model input [OpenAI 2024b]. We use this technique to specify the available project domains the model can use when identifying the project domain, ensuring the model outputs results we expect.
- **Length Specification**: This technique involves specifying the length of the desired output [OpenAI 2024b]. We use this during changeset summarisation to limit the model verbosity.
- **Aligning the Decimals of Numbers**: The comprehension capability of LLMs is likely constrained by the design of BPE tokenisers, which interpret "9.11" as "9", ".", and "11". This limitation is still under discussion [Xie 2024]. Research reveals that this capability can be improved with number formatting [Schwartz et al. 2024]. Since adding special tokens to OpenAI's model is not viable, we used a simple yet effective technique to align the decimals to two digits.

3.1.2 Hyperparameters. Hyperparameters affect the accuracy, verbosity, certainty and more of the LLMs' responses [OpenAI 2024a]. However, due to the high cost of inference, performing rigorous tuning (e.g., grid search) is not an option. Inspired by previous studies [Li et al. 2024; Liu et al. 2024], we greedy searched the following hyperparameters with manual evaluation:

- **Temperature**: The temperature parameter controls randomness, with lower values producing more focused outputs [OpenAI 2024a]. After experiments, we found a temperature of 0 yields consistent and satisfactory outputs, which aligns with previous studies [Peng et al. 2023].
- **Top-p**: Top-p implements nucleus sampling, considering only tokens within the specified probability mass. It controls the balance between certainty and creativity. We found a low top-p value of 0.1 encourages the LLM to generate deterministic output, and reduces the chances of generating casual content in few-shot classification tasks.

3.2 SmartNote Overview

In this section, we outline the stages of SmartNote's generation pipeline and describe its process for generating a release note from a GitHub repository given two versions, the previous version and the updated version. As illustrated in Figure 3, SmartNote is composed of five stages: 1) Info Retriever, 2) Settings Generator, 3) Commit Analyser, 4) Change Summariser, and 5) RN Composer.

In the first stage, information retrieval, SmartNote collects all commit, pull request, and repository data associated with the changes made between the previous and updated versions. Next, in the settings generation stage, SmartNote starts by determining the project domain and commit message quality using the LLM module. Once all the data has been collected and the settings have been determined, SmartNote begins change summarisation. In this stage, commit data is packed into changesets which are then summarised into release note entries with the LLM module. The commit analyser stage is run simultaneously, using the machine learning (ML) classifiers to identify the conventional commit category and significance scores of commits, prerequisites for filtering, summarising, and personalising the release note entries. Finally, in the last stage, SmartNote

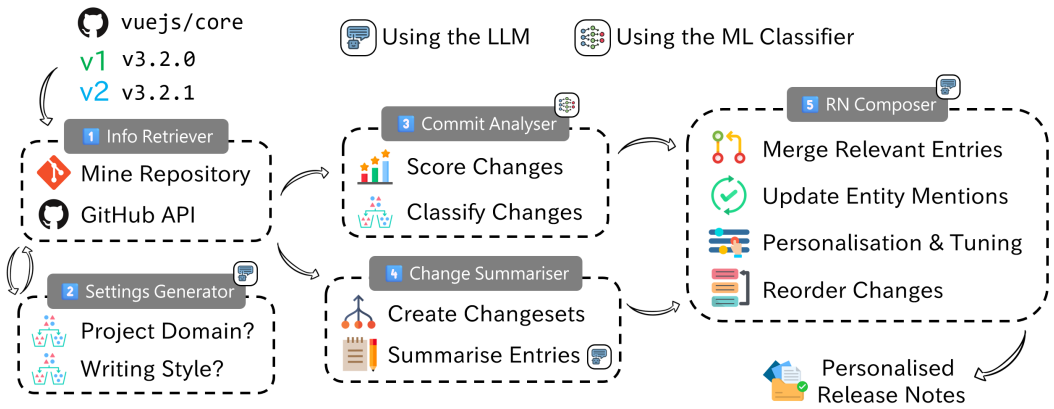


Fig. 3. Overview of the SmartNote Release Note Generation Pipeline

composes the release note. It utilises the LLM module to refine the content by first, rephrasing for conciseness and reorganising the entries, and then by adjusting the release note to align with the release context.

3.3 Info Retriever

In the information retrieval stage, SmartNote compares two versions of a GitHub project and collects data on commits, pull requests, releases and projects. First, SmartNote looks for the corresponding git tags (a reference to a specific point in the repository's history [Git 2024]) of the previous release and the current release, and traverses over the commits with the PyDriller [2024] library. In cases where this approach fails, SmartNote falls back to the GitHub API. Notably, we observed that Conventional Changelog fails when the release points to an orphan commit (e.g., [QuestDB 2024a]), SmartNote is not vulnerable in such cases. Next, SmartNote combines repository mining and the GitHub API to extract: 1) release-related features such as release type (e.g., major, minor, patch), number of commits in the release, number of authors, and number of unique committers; 2) project-related features such as project name, previous versions and new versions, description, and README document; 3) commit-related features such as commit SHA, author, commit date, commit message, file patches (i.e., the code difference between the previous and new version or *git diff*), file extensions, and lines changed; and 4) pull request features such as title, message, and associated commits. Additionally, the information retriever automatically identifies "squash merge" and "rebase merge" pull requests [GitHub 2024] and flags the associated commits — a necessary step for merging individual commits into release note entries.

3.4 Settings Generator

To accommodate the varying needs of projects, releases, and target audiences, SmartNote offers configuration options for project domain, writing style, and structure, based on definitions from previous studies [Wu et al. 2023]. Additionally, it provides options for commit grouping and the minimum significance score (MST). However, configuring these options can be tedious. To simplify this, SmartNote includes a settings generator that is responsible for automatically identifying and applying context-aware defaults based on findings from previous research and heuristics — an approach proven effective in software automation tools [He et al. 2023] — making SmartNote effectively zero-config.

The **project domain** plays a key role in shaping the writing and organisational style of the release note [Moreno et al. 2014; Wu et al. 2023]. SmartNote does not require users to interpret domain definitions and choose the most suitable domain. Instead, it utilises a project domain

classifier that leverages the LLM module to categorise the project domain based on the project's description and README file.

The **writing style** also plays a key role in shaping content and is closely linked to the project domain [Wu et al. 2023]. If the user has not specified a writing style, SmartNote automatically selects one based on the project domain. However, when the expository style is selected — where release note entries are composed of the original commit message — an additional evaluation step is taken to assess the project's suitability for this writing style. If it's unsuitable, the writing style is changed to persuasive. To achieve this, we designed a binary commit message classifier that leverages the LLM module. The prompts are engineered based on findings from a previous study identifying the characteristics of good commit messages [Tian et al. 2022]. The classifier aligns closely with human-labelled data, achieving 95% agreement (Cohen's kappa = 0.9) between two authors who, after carefully reviewing the definitions and examples of "good" commit messages outlined in a previous study [Tian et al. 2022], independently labelled the overall commit message quality for the projects and subsequently discussed their labels to reach consensus.

The **MST** is an option that allows users to specify a minimum commit significance threshold for changes. It enables SmartNote to filter out insignificant changes (e.g., fixing typos in comments or documentation; adjusting white space or indentation; and reformatting code without functional changes like running a linter), ensuring the release note is neither empty nor excessively long. This approach aligns with previous studies [Abebe et al. 2016], which found that well-formed release notes typically include only 6% to 26% of issues addressed in a release. It also reflects developer sentiments [Rahman 2012], which indicate that excessive detail can cause readers to lose attention, reinforcing the importance of conciseness. Therefore, SmartNote's MST has been carefully tuned, ensuring release notes highlight significant changes without overwhelming the reader. This balance is achieved through iterative refinement and heuristic analysis, which determined that an MST between 0.1 and 0.15 strikes an optimal balance. Higher thresholds (0.2 or above) tend to introduce excessive detail, making release notes overly verbose, while lower (0.05 or below) thresholds risk omitting too much information, resulting in sparse release notes. For example, in cases like AkShare v1.14.62 [2024], where most commits are minor or insignificant, increasing the MST helps include more changes. Conversely, in cases like RustDesk v1.3.0 [2024], which has many meaningful commits, lowering the MST ensures a balance between verbosity and conciseness.

The **structure** determines how the release note is formatted. By default, it is set to "Change Type", the most common release note structure in open source projects [Wu et al. 2023]. Lastly **commit grouping** determines whether commits are grouped together. It works by combining commits associated with the same pull request into a single release note entry — an approach commonly seen in real-world projects — using the LLM module, thereby improving conciseness. This setting is enabled by default, as conciseness is critical.

3.5 Commit Analyser

To address the problem of generic and verbose release notes, they are typically structured by change type [Wu et al. 2023], with included changes determined by their perceived importance. However, while developers are categorising changes in release notes, a vast majority of repositories do not categorise commits. Notably, approximately 90% of the projects we sampled do not enforce a standard commit specification, explaining the limited effectiveness of off-the-shelf release note generators that rely solely on parsing commit messages. Moreover, projects varying in domain, scale, and collaboration methods may exhibit distinct development patterns and naming conventions. Accordingly, we design a commit analyser that accounts for variations in content preferences based on release type and project context [Bi et al. 2020; Wu et al. 2023]. The commit analyser performs two tasks: 1) evaluating the relative significance of commits for conciseness and 2) categorising

them for structural organisation. To achieve this, we leverage machine learning classifiers with models that are trained using contextual commit, release, and project features sampled from 3,728 repositories. To train the classifiers, we employ an approach that has been shown to be effective in software engineering studies [Mariano et al. 2019; Xiao et al. 2022]. This involves vectorising the text, combining it with numerical features, and feeding the resulting representation into XGBoost, an efficient, commonly used [Xiao et al. 2022], machine learning classifier. While most contextual features are numerical, LLMs have inherent limitations in processing numerical data effectively [Xie 2024]. In contrast, XGBoost, which implements the gradient boosting decision tree algorithm [Chen and Guestrin 2016], utilises a tree-based decision architecture, improving interpretability and ensuring transparency in the model's decision-making process.

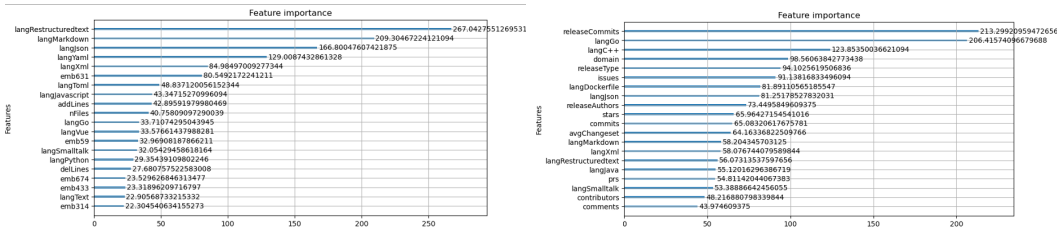
3.5.1 Data Collection. A high-quality dataset is important for creating reliable and precise models. We begin by defining our selection criteria: popular, actively maintained code repositories (i.e., excluding tutorials or resource collections) with a rich development and release history. To this end, we sampled 3,728 non-forked and non-archived repositories on GitHub using the SEART GitHub Search Engine (seart-ghs). These repositories were created before 2020, with more than 5000 stars, more than 10 releases on GitHub, and a codebase exceeding 100 lines. To ensure data quality, we removed six repositories whose git logs could not be parsed, as well as repositories that were duplicated or renamed. Additionally, we removed 23 repositories whose primary language was not English, as multilingual encoding introduces additional complexity and engineering overhead. Finally, we manually verified all repositories, removing five that contained tutorials or configuration files, resulting in a total of 644. The dataset was further processed for commit categorisation by retaining repositories in which at least two-thirds of the commits adhered to the conventional commits specification, resulting in 389 repositories. Lastly, to ensure the commit scoring model captures the intricacies of commit inclusion and exclusion in release notes, we refined the dataset to include only repositories which have more than three links to commits in their release notes, resulting in 272 repositories. From this dataset, we identified 21,882 releases encompassing 715,089 commits, of which 139,423 (17.7%) were explicitly referenced in their corresponding release notes.

3.5.2 Feature Selection. To enhance the accuracy and generalisability of the classifiers, we supply the model with extensive contextual features:

- **Commit Context:** The commit message ($emb*$) is the most significant semantic feature, as high-quality commit messages conclude "what" and "why" of the changes [Tian et al. 2022]. The scale of the changeset, i.e., the number of added lines ($\#addLines$) and deleted lines ($\#delLines$), represents the complexity of the commit and their relative importance [Levin and Yehudai 2017]. Furthermore, we build a programming language identifier using GitHub's linguist library [GitHub-Linguist 2024] to identify the programming languages of files modified in commits ($lang*$), in order to support the commit classifier.
- **Release Context:** The type of release ($releaseType$) influences the content of the release notes [Wu et al. 2023]. We parse and compare release versions with the *semver* library [Preston-Werner 2024] into Major, Minor, Patch, and Unknown (version numbers incompatible with the semantic versioning specification). Moreover, we collect numerical contextual features representing the scale of the release: the number of commits between versions ($\#releaseCommits$), the number of committers and the number of authors ($\#releaseAuthors$). The modification complexity of the release is measured by averaging the number of modified files ($avgChangeset$), lines of code affected per file ($avgCodechurn$), and history complexity ($avgHistoryComplexity$) [Hassan 2009].
- **Project Context:** To quantify the project's scale and complexity, we measure the following metrics at the time of release: the number of commits ($\#commits$), contributors

(*#contributors*), stars (*#stars*), issues (*#issues*), PRs (*#prs*), comments (*#comments*). We categorise the projects into 4 domains (*projectDomain*): *Application Software*, *System Software*, *Libraries and Frameworks*, *Software Tools* derived from the 6 project domains proposed by Borges et al. [2016] and studied by Wu et al. [2023].

Commit messages were encoded into sentence embeddings (fixed-length vectors) leveraging the General Text Embeddings model (`gte-base-en-v1.5`) [Li et al. 2023]. With 137M parameters, it is small yet performant (ranking 31st on the MTEB classification leaderboard [Muennighoff et al. 2022]), and ideal for SmartNote's intended use case — CPU-only CI environments. Statistics were mined with PyDriller, a Python framework that extracts information about commits, developers, modified files, diffs, and source code [Spadini 2024]. For historical events of the repositories, we utilised the GitHub API alongside the GHArchive, a project to record the public GitHub timeline, with the support of ClickHouse, a well-performing OLAP database. Project domains were labelled manually by two authors. The first round of independent labelling yields a Cohen's kappa of 0.57, indicating moderate agreement [McHugh 2012]; Two authors discussed and reached a consensus.



(a) Commit Classification (b) Commit Scoring

Fig. 4. Importance of Features in the Machine Learning Classifiers.

3.5.3 Model Training and Feature Importance. To mitigate the threat of overfitting, we implemented several strategies. First, we randomly partitioned the dataset into training, testing, and validation sets in a 7:2:1 ratio, following prior studies [Jiang et al. 2021; Xiao et al. 2022]. Second, we applied the early stopping technique to terminate training when validation loss began to degrade. Third, we conducted K-Fold validation to ensure the randomness of the split. Finally, we determined the optimal hyperparameters using grid search.

The **commit classification model** converged after 100 iterations, with an accuracy of 0.71, a precision of 0.71, and a recall of 0.71 (numbers are weighted average). The 5-fold validation yielded the same performance numbers. Due to the interpretability of decision trees, the importance of each feature is straightforward, as indicated by XGBoost's reported gain value (the improvement in accuracy brought by a feature to the branches it is on) [XGBoost 2022] in Figure 4. The programming languages of modified files (RST, Markdown, JSON, etc.) stand out, with several dimensions of the sentence embedding also being vital in the model's classification. The **commit scoring model** converged after 100 iterations, with a precision of 0.94 and a recall of 0.94. K-Fold validation yields consistent accuracy numbers between 0.94 and 0.95. Figure 4, shows that the number of commits in the release is the most significant factor in the significance score, while domain, release type, and the number of issues also play key roles for determining commit significance. The project domain and release type are also notable factors, aligning with empirical evidence from previous studies [Bi et al. 2020; Wu et al. 2023].

3.6 Change Summariser

Gathering data from different sources is a common method that's utilised in existing research [Klepper et al. 2016] and ensures that the generated release note contains accurate and reliable

information. In the change summarisation stage, SmartNote combines data from the previous stages into changesets consisting of the commit date & time, author, message, significance, change type, and file patches. The LLM module processes changesets, generating concise and accurate release note entries — typically single sentences or brief paragraphs — that form bullet points in the final release note. By creating a changeset for each commit and summarising it individually, we break down the task into smaller, more manageable sizes for the LLM, preventing hallucinations, and resulting in higher quality summaries. When commit grouping is enabled (which is by default), change summarisation combines all changes associated with a pull request into a changeset, summarises their descriptions, aggregates their scores, and replaces all associated release note entries with a single consolidated entry.

3.7 RN Composer

In the final stage, SmartNote composes the release note by aggregating the changesets into a list using the identified organisation strategy, and then performs several smaller tasks to refine it.

(1) **Merging Relevant Entries:** Occasionally, developers create multiple commits or pull requests for the same feature or change. This can be caused by bugs or alterations to the feature, causing overlapping entries that increase verbosity, decrease clarity, and cause reader confusion. With the exception of release notes with the "Change Priority" structure type, the LLM module is utilised to remedy this by merging related entries, consolidating them to improve clarity and readability, while ensuring no information is lost. E.g., in version 1.3.1 of Sniffnet [2024], there were 22 commits referencing translation. In the release note produced by SmartNote, these were consolidated into 10 entries.

(2) **Updating Entity Mentions:** In major refactoring releases, it's common for an entity (a function or a variable) to be added, renamed, or removed. This causes fragmented entries that refer to the same entity as it worked or was named in different points of time in the past. To address this inconsistency, SmartNote utilises the LLM module to identify renamed or modified entities and updates their mentions in the release note to match the most recent repository state at the specified version, improving coherence and clarity. E.g., in PR #109 of the d3 project [2024], when a function was renamed based on feedback from the project maintainer.

(3) **Personalisation and Tuning:** To improve conciseness, SmartNote removes details based on the content preferences of the project and its audience. Specifically, entries with lower significance scores than the specified MST are removed; and the release note is trimmed and summarised using the LLM based on the project domain's description and content preferences, guided by findings in [Wu et al. 2023]. E.g., AKShare v1.14.62 [2024], which includes a single, simple code change. Without this step, a verbose 36-line release note is generated, with this step, changes are condensed into a single entry.

(4) **Reordering Changes:** The inverted pyramid principle of writing suggests placing the most fundamental information at the top. Ordering with predefined rules may suffer from low generalisability though, as a previous study reveals that different projects prioritise different changes [Wu et al. 2023]. SmartNote utilises the LLM to reorder the categories, moving breaking changes and new features, bug fixes, and enhancements to the top as they are considered more important [Abebe et al. 2016; Wu et al. 2023]. Document changes, dependency updates, and version changes are considered less important and placed lower. E.g., in Bevy v0.14.1 [2024], features are moved to the top.

4 Evaluation

SmartNote is designed to utilise context-awareness to generate high-quality, personalised release notes whilst being broadly applicable. To this end, we design our evaluation to answer three

questions: 1) Does SmartNote generate high-quality release notes? — necessary to understand in which aspects SmartNote advances in; 2) Is SmartNote's personalisation method effective? — to understand SmartNote's personalisation capabilities and effectiveness; and 3) How applicable is SmartNote? — to compare against existing tools and understand if there are any applicability limitations.

Table 1. Distribution of project domains among the projects for evaluation.

Project Domain	Total Projects	% of Projects
Application Software	7	30.43%
Libraries & Frameworks	6	26.09%
Software Tool	5	21.74%
System Software	5	21.74%
Total	23	100%

To ensure a comprehensive comparison, we collected feedback for four types of release notes: first, for **SmartNote**; second, for ***DeepRelease***, a state-of-the-art alternative, recent deep-learning power generator with accessible data; third, for the **original release notes** produced by the maintainers; and finally, for ***Conventional Changelog***, a remarkably popular off-the-shelf tool with more than one million downloads per week on NPM, a JavaScript package manager [NPM 2024]. This approach covers a wide variety of release notes, enabling us to effectively compare our work against existing methods and solutions.

We began by brainstorming emerging open source projects from the GitHub trending repositories page [2025], considering programming languages such as Python, Rust, C++, C#, Java, JavaScript, and TypeScript. We selected 33 projects with over 500 stars, under active development, and with recent community engagement (i.e., new commits, issues, discussions, or release history) and which allow for issues or discussions of enhancements, features or feedback. Another author independently reviewed the selected projects, ensuring they meet the criteria and identifying the project domain. The authors discussed any inconsistencies to reach a consensus. The projects were reevaluated for a final time, and those that were historical, extremely large, seldom publish releases, or appear to follow organisational guidelines to produce release notes were excluded as they are unlikely to be interested in our study. Furthermore, we performed under-sampling to balance the representation of each domain. After reevaluation, 10 projects were removed, resulting in a total of 23 projects, as shown in Table 1. Next, we obtained their latest releases (7 minor releases and 16 patch releases) and release notes from the GitHub API. Finally, we generate release notes using SmartNote, DeepRelease, and Conventional Changelog. Note that we used the default and automatic options of SmartNote, to avoid overwhelming survey participants with multiple variations of the release note. Additionally, we calculated the cost for SmartNote to generate release notes for the evaluation projects with automatic settings. SmartNote's automatic release note generation costs an average of 90 cents per release ($20.82 for 23 releases), which is economical given the extensive time typically required for high-quality release notes [Moreno et al. 2014].

4.1 EQ1: Does SmartNote Generate High-Quality Release Notes?

We conducted a human and an automatic evaluation to assess the effectiveness of SmartNote and the release notes generated by it.

4.1.1 Quality Expectations. To measure release note content quality, we identify four key criteria from previous work:

(1) **Completeness**, to understand whether the release note sufficiently covers the changes made between the analysed versions. Missing or incorrect change information is a major issue in release note practices [Wu et al. 2022] and software documentation [Aghajani et al. 2020].
(2) **Clarity**, to understand whether the release note accurately and understandably reflects the changes made in the project. The absence of clarity has been identified as a central issue in software documentation by previous studies [Aghajani et al. 2020]. The changes need to be explained with rationales and details to be understandable by target users.
(3) **Conciseness**, to understand whether the release note succinctly provides the right amount of information. Studies confirm that conciseness is important [Aghajani et al. 2020]. However, unnecessarily verbose release notes scare readers [Rahman 2012] which explains why most release notes only list between 6%-26% of issues addressed in a release [Abebe et al. 2016].
(4) **Organisation**, to understand whether the release note is structured clearly and logically. Practitioners prefer release notes to contain many different aspects of information and are organised into categories [Wu et al. 2022]. Well-formed release notes positively impact software evolution [Bi et al. 2020].

Table 2. Human and automatic evaluation results for release notes.

Criteria	SmartNote	DeepRelease	Original	Conventional Changelog
Completeness	**4.00 (81%)**	3.39 (61%)	3.71 (74%)	2.74 (45%)
Clarity	**4.06 (84%)**	2.97 (42%)	3.81 (77%)	2.71 (45%)
Conciseness	3.35 (55%)	3.03 (42%)	**3.68 (65%)**	2.52 (29%)
Organisation	**4.10 (84%)**	3.42 (55%)	3.52 (61%)	2.61 (35%)
Commit Coverage	**81%**	41%	31%	13%
Information Entropy	**1.59**	1.04	0.78	0.45
Token Count	1162.48	**251.70**	498.52	2231.57
Automated Readability Index	33.06	**26.19**	30.93	102.38
Dale-Chall Readability	**13.35**	16.22	13.79	32.70
Entity Percentage	1%	7%	2%	4%
Success Rate	**23 (100%)**	21 (90.48%)	N/A	15 (46.67%)

4.1.2 Human Evaluation. To conduct the human evaluation, we created a unique questionnaire for each project, allowing us to gather feedback from users most familiar with it and its domain. Using 5-point Likert scale questions, the questionnaire was designed to assess the four criteria identified earlier. Following this, we recruit participants with diverse backgrounds for the credibility of the survey. First, we reached out to the most active maintainers of the projects via their public contact information. We received one response from a maintainer who was willing to participate in the survey. Next, we utilised snowball sampling to recruit ten open-source developers and six researchers with experience ranging from less than 1 year to over 8 years. In total, we recruited 17 participants, with the most common level of experience at 2 to 4 years, representing 55% of participants followed by 4 to 8 years representing 29% of participants. Participants selected up to 3 projects based on their familiarity, checked the projects' GitHub release pages, and completed questionnaires. The release note generator's name was masked to prevent bias. Once a project received 3 responses, we removed it from the list, spreading the responses across all the projects. Table 2 presents the survey results, with scores averaged across all projects. The agreement percentage, shown in brackets, indicates the proportion of participants who either agreed or strongly agreed.

SmartNote outperforms all other tools in the categories of completeness, clarity, and organisation while conciseness achieves the second best performance.

The release notes for all evaluated projects are available in our replication package. As a glimpse, we highlight projects where SmartNote rated significantly better, slightly better, or worse:

- **Significantly Better**: AkShare (Figure 1), LangChain, Sniffnet, and UniGetUI. In these projects, SmartNote produced release notes that were organised, context-aware, audience-specific, and prioritised significant changes while filtering out irrelevant details compared to raw LLM outputs and other tools. For example, the AKShare project where the commit messages and PRs are simple, SmartNote addresses the changes understandably and concisely through code comprehension; in contrast, other tools borrow the developers' words and output nonsense.
- **Slightly Better**: Zulip, StirlingPDF, es-toolkit, and Continue. These projects demonstrated improvements in clarity and organisation. SmartNote grouped related changes and provided well-structured summaries. Other tools and the raw LLM are likely to benefit from the fruitful commit messages and PRs in these projects and generate release notes of acceptable quality. While the improvements were less dramatic, the notes were still easier to interpret and more actionable than those generated by baseline tools.
- **Worse**: Jan is likely rated worse for the amount of dropped details (SmartNote: 2 entries vs Conventional Changelog: 16 entries). The default MST configuration may have excluded changes that some users consider relevant.

The results of the survey reflect the performance of the fully automated pipeline. Considering this, SmartNote outperforms both the original release notes and competing solutions in terms of completeness, clarity, and organisation. Over 80% of participants agree that SmartNote performs the best among the compared tools. While only 55% agree regarding conciseness, this is unsurprising given our evaluation strategy. With additional project-specific tuning, users can achieve better results. In light of this, we can confidently say that SmartNote is significantly superior to off-the-shelf tools and state-of-the-art alternatives.

4.1.3 Automatic Evaluation. Although human evaluation is undoubtedly the best way to assess release notes, we conducted an automatic evaluation to 1) complement our findings; and 2) explore the possibility of automatic quality assessment on RNs. Previous work [Jiang et al. 2021; Li et al. 2024; Zhang et al. 2024] employed text similarity metrics such as BLEU and ROUGE for the purpose of comparing and measuring the differences between automatically generated release notes and gold standards. However, we find their approach to unviable for two reasons. First, previous studies that applied BLEU or ROUGE on generated release notes did not publish their evaluation dataset so it's impossible to compare. Second, text similarity metrics compare the lexical similarity of the text against "gold references"; they can penalise semantically correct hypotheses if they differ lexically from the reference [Wieting et al. 2019]. Thus, they are likely sub-optimal for the task, given the length and diversity of high-quality release notes.

To address this, we designed six metrics to automatically measure the four quality aspects of our study. **First**, we used commit coverage to assess **completeness**, calculated by dividing the number of mentioned commits by the total number of commits in the release. A higher value indicates better overall completeness. **Second**, we measured **conciseness** using the token count of OpenAI's tokeniser, as the word count doesn't play well with code snippets. **Third**, to evaluate **organisation**, we leveraged markdown parsing to identify categories (headers) and items (bullet points), and we calculated information entropy, where a higher value signifies a higher number of categories and a more balanced distribution of entries, indicating better organisation. **Finally**, for **clarity**, we aimed to assess both specificity and understandability. We measured specificity by

calculating the density of entities (specific software engineering terms, like the names of operating systems and libraries) with a specific software engineering named entity recogniser [Nguyen et al. 2023], where a higher value suggests more technical details, and a neutral value is optimal. For understandability, we used the Automated Readability Index [Smith and Senter 1967] calculated the average number of characters per word and of the words per sentence, where a lower score is better; and the Dale-Chall readability formula [Dale and Chall 1948] to measure word commonality, where lower scores also indicate better readability.

Results in Table 2 show that, compared to the next best alternative, SmartNote achieves twice the performance with 81% coverage, whereas the original release notes achieve only 31% coverage. In terms of token count, which reflects conciseness, SmartNote ranks third; however, this is not a concern, as users can easily adjust the level of conciseness through the configuration settings. These results suggest that release note authors tend to sacrifice coverage for conciseness, as indicated by the token count — the number of words in the release note. In terms of organisation, SmartNote ranks first, with an entropy of 1.59, a significant improvement over DeepRelease, the next best tool, which has an entropy of 1.04. This indicates that SmartNote organises information much more effectively, consistent with the findings from our human evaluation. Finally, the automatic metrics for clarity yield mixed findings, SmartNote ranks third for the automated readability index but first for the Dale-Chall readability score, while having the lowest entity percentage, which measures specificity.

Overall, SmartNote demonstrates superior organisation and provides significantly better commit coverage compared to baseline methods. These findings align with human evaluations, further confirming that SmartNote outperforms off-the-shelf tools and state-of-the-art alternatives. In contrast, SmartNote's rank for conciseness suggests that the default settings we used may not produce ideal results. However, the results highlight that release note authors make sacrifices in order to compensate for other aspects, e.g., compromising on commit coverage to improve conciseness. Therefore, reinforcing the importance of giving users the control to make adjustments where necessary based on their preferences. This demonstrates that SmartNote is adaptable to both user and project needs, providing a significant advantage over static, off-the-shelf solutions.

Result of EQ1: The human evaluation shows over 80% of participants agree that SmartNote performs best in completeness, clarity, and organisation. While only 55% agree regarding conciseness. The automatic evaluation shows that SmartNote ranked first in completeness with 81% commit coverage and organisation with an information entropy score of 1.59. Conciseness ranked third with a token count of 1162.48 and clarity which showed mixed findings. Overall, the automated evaluation aligns with the human evaluation. Conciseness did not achieve superior performance in either, but this is a non-issue as SmartNote offers the flexibility to customise conciseness as preferred.

4.2 EQ2: How Applicable is SmartNote?

To ensure applicability, it is important that release note generators can be used with all projects. Existing tools have stringent requirements, such as commit convention, templates, labels, or PR strategies; require extensive configuration; and some work only for certain programming languages. e.g., ARENA. SmartNote addresses these limitations by: 1) using a settings generator to determine optimal configurations, 2) ensuring developers do not have to follow any requirements, and 3) by being language-agnostic. Moreover, as shown in Figure 1, **SmartNote** adapts to the size and complexity of each project, producing concise and relevant release notes even for small projects. To this extent, we analyse the projects for which we generated release notes for to identify which succeeded and which failed (i.e., generating empty or completely wrong release notes).

The success rate of each project is recorded in Table 2 and shows that SmartNote is widely applicable. Compared to existing tools, SmartNote does not fail at generating release notes for any project, while DeepRelease and Conventional Changelog fail approximately 10% and 54% of the time respectively. These failures can be attributed to the several main limitations. In the case of DeepRelease, it is only able to process PRs (e.g., bevy between version v0.14.0 and v0.14.1 [Engine 2024]). While in the case of Conventional Changelog, it is unable to process off-tree commits (e.g., questdb [QuestDB 2024a]), i.e., changes that are not between the two specified versions. Also, in cases where there's a lack of labelling and commit conventions, it does not produce any output (e.g., flatbuffers between version v24.3.7 and v24.3.25 [Google 2024]).

Result of EQ2: SmartNote successfully generates release notes for all projects, while DeepRelease fails for approximately 10% of projects due to its stringent PR requirements and Conventional Changelog fails approximately 54% of the time due its rigid commit convention requirement.

4.3 EQ3: How Effective is SmartNote's Personalisation?

To better understand how SmartNote's contextually aware, personalised generation pipeline contributes to the quality of release notes, we conducted an ablation study with both automatic and human evaluations, comparing SmartNote's release note with three different variants:

- **Raw LLM**: To understand the contribution of the LLM, we instruct it to generate a release note without any prompt engineering, guidelines or examples, relying solely on its world knowledge and comprehension capabilities.
- **No Composer**: A release note generated by SmartNote without the RN Composer stage, a key component of its personalisation capabilities.
- **Random Context**: A release note generated by SmartNote with a randomly selected project domain and an "Unknown" release type.

For the human evaluation, we recruited a separate group of six participants, consisting of students and industry professionals, to rank release notes based on the four key aspects previously discussed: completeness, clarity, conciseness, and organisation. To mitigate author bias between the previous survey and the ablation, we asked participants in both groups to score the generated release note for both the **Raw LLM** and the **Random Context** variant. Furthermore, to ensure a fair comparison, we applied weighted normalisation to our results, shown in Table 3.

The results of the human evaluation for the **Raw LLM** variant show that participants consistently rated it lower across all metrics: completeness (4.00 vs. 2.90), clarity (4.06 vs. 3.12), conciseness (3.35

Table 3. Human and automatic evaluation results for the ablation study.

Criteria	Raw LLM	No Composer	Random Context	SmartNote
Completeness	2.90 (41%)	3.21 (54%)	3.31 (55%)	**4.00 (81%)**
Clarity	3.12 (57%)	3.31 (61%)	3.28 (55%)	**4.06 (84%)**
Conciseness	2.91 (45%)	3.18 (53%)	3.11 (43%)	**3.35 (55%)**
Organisation	3.04 (51%)	3.40 (63%)	3.87 (83%)	**4.10 (84%)**
Commit Coverage	30%	79%	**82%**	81%
Information Entropy	**2.04**	1.21	1.42	1.59
Token Count	**568.35**	1029.22	970.17	1162.48
Automated Readability Index	**14.46**	33.22	30.44	33.06
Dale-Chall Readability	**11.01**	13.50	13.89	13.35
Entity Percentage	3%	1%	2%	1%

vs. 2.91), and organisation (4.10 vs. 3.04), confirming the significance of our prompt engineering efforts. Further examination revealed: 1) Inconsistent Styles and Structures: E.g., a long, plain list of pull request titles and files changed after the organised list of changes for Manticore-Search v6.3.4 [2024]. 2) Verbose Results for Simple Changes: E.g., a verbose 36-line release note for AKShare v1.14.62 [2024], despite the change involving a single, simple code modification. 3) Minimal Technical Details and Practical Implications: E.g., "various code quality improvements and refactoring for better maintainability and readability" in a release note generated for QuestDB v8.1.0 [2024b]. In comparison, the automatic evaluation yields mixed results. The commit coverage is lower and aligns with the human evaluation well, explaining why the token count is considerably lower when compared to other variants — the LLM is sacrificing coverage for brevity. While initially, the information entropy indicates good organisation, we observe that an excessive information entropy score is not ideal in real world applications, due to its inconsistent categorisation and excessive granularity. E.g., in Zulip v9.1 [2024] where most commits would traditionally be categorised as documentation updates.

Next, we examine the human evaluation for the **No Composer** and **Random Context** variants. As shown in Table 3, they similarly exhibit lower completeness and clarity, confirming that contextual understanding plays a significant role in generating more comprehensive and clearer release notes. Conciseness, however, presents a similar pattern to SmartNote. While it's relatively lower across all variants, it still outperforms most baselines except for original, human written release notes (65%), suggesting that release note authors tend to prioritise brevity by omitting details. These results indicate that the MST needs to be fine-tuned on a project-by-project basis to achieve balance between commit coverage and conciseness. On the other hand, organisation remains one of SmartNote's strongest aspects, with the No Composer variant, the worst performing one, still achieving 63%, even without full contextual understanding. We attribute this to the LLMs world knowledge and code comprehension capabilities. In contrast, the automatic evaluation results indicate that the No Composer and Random Context variants perform comparably to SmartNote. However, they do not align with the human evaluation, which indicates that while the release notes perform well in automated metrics, they fail to capture the nuances of human-written release notes, as seen in SmartNote, which benefits from prompt engineering.

Result of EQ3: To assess the impact of context awareness on release note quality, we conducted an ablation study with both automatic and human evaluations, comparing SmartNote's generated release note with three variants: Raw LLM (simply feed the changes to an LLM without any prompt engineering, guidelines or examples), No Composer (without the composition stage), and Random Context (random project domain). The human evaluation of the Raw LLM, No Composer, and Random Context variants revealed that they are overly verbose and inconsistent. The automatic evaluation for the Raw LLM aligns with these finding. However, while the automatic evaluation of the No Composer and Random Context variants indicated metrics within margin of SmartNote, they miss the nuance of human-written release notes, captured by SmartNote, which is attributed to prompt engineering and context awareness. In summary, these findings highlight that context comprehension is key to SmartNote's effectiveness, particularly in completeness and clarity.

5 Limitations

In this section, we discuss and address the limitations of our study, highlighting factors that may affect the validity of our study to guide future research. To this extent, we cover two factors: 1) internal validity, and 2) external validity.

5.1 Internal Validity

This concerns factors that are internal to the study which could potentially affect the study from accurately measuring what it intends to. Manually labelling the project domains in the dataset for training the classifiers may introduce author bias. To mitigate this, two authors independently examined and labelled the data, and inconsistencies were discussed and resolved with a consensus. Moreover, the variability of classifier accuracy across different projects (e.g., lower accuracy in projects with simple changes) could impact the results of the evaluation. For most projects in the evaluation set, the classifier's accuracy is approximately 0.6, which is acceptable given that a random classifier would achieve less than 0.1. However, for repositories where changes are simple and maintainers are not inclined to write meaningful release notes (e.g., AgentGPT), the classifier's accuracy can drop to around 0.3. To address this, we employ a range of different projects in our evaluation and recommend maintainers utilise commit message standards or prevent writing confusing and shorthand commit messages. Automated commit message generators [Li et al. 2024; Zhang et al. 2024] can be employed to enhance the quality of commit messages and release notes. Additionally, the assessment of the release notes in our survey may be influenced by personal preferences and experience, which could affect the objectivity of the evaluation. This challenge is exacerbated by the absence of standardised evaluation criteria, potentially introducing bias into subjective judgements. To address this, a wide range of industry developers and PhD students were invited to participate in evaluations, thereby aiming to reduce bias through varied perspectives and enhance the overall objectivity and credibility of the assessment.

5.2 External Validity

This concerns the generalisability of the findings of this study. First, SmartNote has only been tested on OpenAI's "gpt-4o" model. However, the results of this paper should be generalisable to other top-tier LLMs (e.g., Claude, Gemini, Qwen) with another round of prompt engineering. We were not able to do so because of the high cost of LLM inference. Second, the human evaluation is performed on a relatively small evaluation set of 23 open-source projects, which may cast doubts on generalisability. However, the scale of evaluation is limited by the constraint of developer hours and communication efforts required to conduct the survey. Additionally, we ensured a diverse range of projects and participants were represented: projects from varying domains, sizes, and popularity, and participants of various backgrounds were involved in the study. Moreover, many off-the-shelf tools have features and quality-of-life aspects that maintainers may have become accustomed (e.g., first-time contributors). The absence of these features decreases the broad applicability of SmartNote.

6 Conclusion

In response to the limitations of existing release note generators and off-the-shelf solutions, this study introduces SmartNote, a novel, contextually tailored automated release note generation approach designed for broad applicability and personalisation. SmartNote leverages the latest in LLM technology and insights from empirical research to personalise to project domains and generate high-quality release notes without imposing stringent requirements or necessitating adjustments to existing workflows. In future research, we aim to implement and test SmartNote in real-world scenarios to validate its effectiveness and contribute to the open-source community by building a tool with interactive editing capabilities that integrates into the release pipeline.

7 Data Availability

The dataset and source code for this study are publicly available in a replication package on Figshare [2025]. For the complete codebase, setup instructions, and additional resources, please visit the GitHub repository: https://github.com/osslab-pku/SmartNote.

Acknowledgments

This work is supported by the National Natural Science Foundation of China Grant 61825201 and 62142201.

References

Surafel Lemma Abebe, Nasir Ali, and Ahmed E Hassan. 2016. An empirical study of software release notes. *Empirical Software Engineering* 21 (2016), 1107–1142.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2024. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL] https://arxiv.org/abs/2303.08774

Emad Aghajani, Csaba Nagy, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, Michele Lanza, and David C Shepherd. 2020. Software documentation: the practitioners' perspective. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 590–601. https://doi.org/10.1145/3377811.3380405

AKFamily. 2024. *Comparing release-v1.14.61...release-v1.14.62 · akfamily/akshare · GitHub.* Retrieved September 13, 2024 from https://github.com/akfamily/akshare/compare/release-v1.14.61...release-v1.14.62

Abdulkareem Alali, Huzefa Kagdi, and Jonathan I Maletic. 2008. What's a typical commit? A characterization of open source software repositories. In *2008 16th IEEE International Conference on Program Comprehension*. IEEE, 182–191. https://doi.org/10.1109/ICPC.2008.24

Anthropic. 2024. Claude 3.5 Sonnet. https://www.anthropic.com/news/claude-3-5-sonnet.

Ray Bernard. 2012. *Convergence Q&A: The Answer is in Black and White | Security Info Watch.* Retrieved August 13, 2024 from https://www.securityinfowatch.com/cybersecurity/article/10840073/the-importance-of-release-notes

Tingting Bi, Xin Xia, David Lo, John Grundy, and Thomas Zimmermann. 2020. An empirical study of release note production and usage in practice. *IEEE Transactions on Software Engineering* (2020). https://doi.org/10.1109/TSE.2020.3038881

Hudson Borges, Andre Hora, and Marco Tulio Valente. 2016. Understanding the Factors That Impact the Popularity of GitHub Repositories. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 334–344. https://doi.org/10.1109/ICSME.2016.31

Tom B Brown. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

Banghao Chen, Zhaofeng Zhang, Nicolas Langrené, and Shengxin Zhu. 2024. Unleashing the potential of prompt engineering in Large Language Models: a comprehensive review. arXiv:2310.14735 [cs.CL] https://arxiv.org/abs/2310.14735

Lianping Chen. 2015. Continuous delivery: Huge benefits, but challenges too. *IEEE software* 32, 2 (2015), 50–54.

Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi (Eds.). ACM, 785–794. https://doi.org/10.1145/2939672.2939785

D3. 2024. *GitHub - d3/d3: Bring data to life with SVG, Canvas and HTML.* Retrieved September 13, 2024 from https://github.com/d3/d3

Edgar Dale and Jeanne S Chall. 1948. A formula for predicting readability: Instructions. *Educational research bulletin* (1948), 37–54.

Farbod Daneshyan, Runzhi He, Jianyu Wu, and Minghui Zhou. 2025. *Replication Package for SmartNote: An LLM-powered, Personalised Release Note Generator That Just Works.* https://doi.org/10.6084/m9.figshare.26994352.v2

Sabit Ekin. 2023. Prompt engineering for ChatGPT: a quick guide to techniques, tips, and best practices. *Authorea Preprints* (2023).

Bevy Engine. 2024. *Comparing v0.14.0...v0.14.1 · bevyengine/bevy.* Retrieved September 13, 2024 from https://github.com/bevyengine/bevy/compare/v0.14.0...v0.14.1

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

Git. 2024. *2.6 Git Basics - Tagging.* Retrieved August 15, 2024 from https://git-scm.com/book/en/v2/Git-Basics-Tagging

GitHub. 2024. *About pull request merges.* Retrieved August 13, 2024 from https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/incorporating-changes-from-a-pull-request/about-pull-request-merges

GitHub. 2025. *Trending repositories on GitHub today.* Retrieved January 31, 2025 from https://github.com/trending

GitHub-Linguist. 2024. *github-linguist/linguist: Language Savant. If your repository's language is being reported incorrectly, send us a pull request!* Retrieved March 27, 2024 from https://github.com/github-linguist/linguist

Google. 2024. *Comparing v24.3.7...v24.3.25 · google/flatbuffers.* Retrieved September 13, 2024 from https://github.com/google/flatbuffers/compare/v24.3.7...v24.3.25

Jasmine Greenaway. 2018. *How to automate your release notes.* Retrieved August 09, 2024 from https://opensource.microsoft.com/blog/2018/09/06/how-to-automate-software-release-notes/

GyulyVGC. 2024. *Comparing v1.3.0...v1.3.1 · GyulyVGC/sniffnet.* Retrieved January 21, 2025 from https://github.com/GyulyVGC/sniffnet/compare/v1.3.0...v1.3.1

A. Hassan. 2009. Predicting faults using the complexity of code changes. *2009 IEEE 31st International Conference on Software Engineering* (2009), 78–88.

Runzhi He, Hao He, Yuxia Zhang, and Minghui Zhou. 2023. Automating Dependency Updates in Practice: An Exploratory Study on GitHub Dependabot. *IEEE Trans. Softw. Eng.* 49, 8 (aug 2023), 4004–4022. https://doi.org/10.1109/TSE.2023.3278129

Jez Humble and David Farley. 2010. *Continuous delivery: reliable software releases through build, test, and deployment automation.* Pearson Education.

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 (APSEC)*. 101–110. https://doi.org/10.1109/APSEC53868.2021.00018

Andrew Joslin. 2024. *conventional-changelog/conventional-changelog.* Retrieved August 13, 2024 from https://github.com/conventional-changelog/conventional-changelog

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)*. 8718–8735.

Foutse Khomh, Bram Adams, Tejinder Dhaliwal, and Ying Zou. 2015. Understanding the impact of rapid releases on software quality: The case of firefox. *Empirical Software Engineering* 20 (2015), 336–373.

Sebastian Klepper, Stephan Krusche, and Bernd Brügge. 2016. Semi-automatic Generation of Audience-Specific Release Notes. In *2016 IEEE/ACM International Workshop on Continuous Software Evolution and Delivery (CSED)*. 19–22. https://doi.org/10.1145/2896941.2896953

Petr Korolev. 2024. *github-changelog-generator.* Retrieved August 13, 2024 from https://github.com/github-changelog-generator/github-changelog-generator

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*. ACM, 97–106.

Jiawei Li, David Faragó, Christian Petrov, and Iftekhar Ahmed. 2024. Only diff Is Not Enough: Generating Commit Messages Leveraging Reasoning and Action of Large Language Model. *Proc. ACM Softw. Eng.* 1, FSE, Article 34 (jul 2024), 22 pages. https://doi.org/10.1145/3643760

Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. 2023. Towards General Text Embeddings with Multi-stage Contrastive Learning. arXiv:2308.03281 [cs.CL] https://arxiv.org/abs/2308.03281

Jiangming Liu, Matt Gardner, Shay B. Cohen, and Mirella Lapata. 2020. Multi-Step Inference for Reasoning Over Paragraphs. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, Online, 3040–3050. https://doi.org/10.18653/v1/2020.emnlp-main.245

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2024. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems* 36 (2024).

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. *ACM Comput. Surv.* 55, 9, Article 195 (Jan. 2023), 35 pages. https://doi.org/10.1145/3560815

LMSYS and UC Berkeley SkyLab. 2024. *LMSYS Chatbot Arena Leaderboard.* Retrieved September 12, 2024 from https://lmarena.ai/

Tim Lucas. 2024. *Release-Drafter.* Retrieved August 13, 2024 from https://github.com/release-drafter/release-drafter

Walid Maalej and Hans-Jörg Happel. 2010. Can development work describe itself?. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. IEEE, 191–200. https://doi.org/10.1109/MSR.2010.5463344

manticoresearch. 2024. *Comparing 6.3.2...6.3.4 · manticoresoftware/manticoresearch.* Retrieved February 10, 2025 from https://github.com/manticoresoftware/manticoresearch/compare/6.3.2...6.3.4

Steve Mao. 2024. *Conventional Commits.* Retrieved August 13, 2024 from https://www.conventionalcommits.org/

Richard VR Mariano, Geanderson E dos Santos, Markos V de Almeida, and Wladmir C Brandão. 2019. Feature changes in source code for commit classification into maintenance activities. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*. IEEE, 515–518.

Mary L McHugh. 2012. Interrater reliability: the kappa statistic. *Biochemia medica* 22, 3 (2012), 276–282.

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*. ACM, 484–495.

Laura Moreno, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Andrian Marcus, and Gerardo Canfora. 2017. ARENA: An Approach for the Automated Generation of Release Notes. *IEEE Transactions on Software Engineering* 43, 2 (2017), 106–127. https://doi.org/10.1109/TSE.2016.2591536

Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. 2022. MTEB: Massive Text Embedding Benchmark. *arXiv preprint arXiv:2210.07316* (2022). https://doi.org/10.48550/ARXIV.2210.07316

Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2024. Using an LLM to Help With Code Understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (Lisbon, Portugal) *(ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 97, 13 pages. https://doi.org/10.1145/3597503.3639187

Sristy Sumana Nath and Banani Roy. 2021. Towards Automatically Generating Release Notes using Extractive Summarization Technique. In *International Conference on Software Engineering & Knowledge Engineering, SEKE*. 241–248.

Sristy Sumana Nath and Banani Roy. 2022. Exploring Relevant Artifacts of Release Notes: The Practitioners' Perspective. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 1270–1277. https://doi.org/10.1109/SANER53432.2022.00152

Sristy Sumana Nath and Banani Roy. 2024. Practitioners' expectations on automated release note generation techniques. *Journal of Software: Evolution and Process* 36, 8 (2024), e2657.

Tai Nguyen, Yifeng Di, Joohan Lee, Muhao Chen, and Tianyi Zhang. 2023. Software Entity Recognition with Noise-Robust Learning. In *Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE'23)*. IEEE/ACM.

NPM. 2024. *conventional-changelog - npm*. Retrieved August 13, 2024 from https://www.npmjs.com/package/conventional-changelog

OpenAI. 2024a. *API Reference - OpenAI API*. Retrieved August 17, 2024 from https://platform.openai.com/docs/api-reference/chat

OpenAI. 2024b. *Prompt engineering - OpenAI API*. Retrieved August 13, 2024 from https://platform.openai.com/docs/guides/prompt-engineering

OpenStack. 2022. *Release Management — OpenStack Project Team Guide documentation*. Retrieved September 02, 2024 from https://docs.openstack.org/project-team-guide/release-management.html#when-to-add-release-notes

Pooya Parsa. 2024. *changelogen*. Retrieved August 13, 2024 from https://github.com/unjs/changelogen

Keqin Peng, Liang Ding, Qihuang Zhong, Li Shen, Xuebo Liu, Min Zhang, Yuanxin Ouyang, and Dacheng Tao. 2023. Towards Making the Most of ChatGPT for Machine Translation. arXiv:2303.13780 [cs.CL] https://arxiv.org/abs/2303.13780

Tom Preston-Werner. 2024. *python-semver/python-semver: Python package to work with Semantic Versioning*. Retrieved March 27, 2024 from https://github.com/python-semver/python-semver

QuestDB. 2024a. *Commit 412f81b - questdb/questdb*. Retrieved February 10, 2024 from https://github.com/questdb/questdb/commit/412f81b337a88478751ce94c95504a6b4b67c29b

QuestDB. 2024b. *Comparing 8.0.3...8.1.0 · questdb/questdb*. Retrieved February 10, 2025 from https://github.com/questdb/questdb/compare/8.0.3...8.1.0

Mahbubur Rahman. 2012. *Good Practices of writing release notes*. Retrieved September 12, 2024 from https://softwareengineering.stackexchange.com/questions/167578/good-practices-of-writing-release-notes

Laria Reynolds and Kyle McDonell. 2021. Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI EA '21)*. Association for Computing Machinery, New York, NY, USA, Article 314, 7 pages. https://doi.org/10.1145/3411763.3451760

RustDesk. 2024. *Comparing 1.2.7...1.3.0 · rustdesk/rustdesk*. Retrieved February 11, 2025 from https://github.com/rustdesk/rustdesk/compare/1.2.7...1.3.0

Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. 2024. A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint arXiv:2402.07927* (2024).

Ted Sanders and Simón Fishman. 2023. *Related resources from around the web | OpenAI Cookbook*. Retrieved August 13, 2024 from https://cookbook.openai.com/articles/related_resources

Sander Schulhoff, Michael Ilie, Nishant Balepur, Konstantine Kahadze, Amanda Liu, Chenglei Si, Yinheng Li, Aayush Gupta, HyoJung Han, Sevien Schulhoff, Pranav Sandeep Dulepet, Saurav Vidyadhara, Dayeon Ki, Sweta Agrawal, Chau Pham, Gerson Kroiz, Feileen Li, Hudson Tao, Ashay Srivastava, Hevander Da Costa, Saloni Gupta, Megan L. Rogers, Inna Goncearenco, Giuseppe Sarli, Igor Galynker, Denis Peskoff, Marine Carpuat, Jules White, Shyamal Anadkat, Alexander Hoyle, and Philip Resnik. 2024. The Prompt Report: A Systematic Survey of Prompting Techniques. arXiv:2406.06608 [cs.CL] https://arxiv.org/abs/2406.06608

Sander Schulhoff, Jeremy Pinto, Anaum Khan, Louis-François Bouchard, Chenglei Si, Svetlina Anati, Valen Tagliabue, Anson Kost, Christopher Carnahan, and Jordan Boyd-Graber. 2023. Ignore This Title and HackAPrompt: Exposing Systemic Vulnerabilities of LLMs Through a Global Prompt Hacking Competition. In *Proceedings of the 2023 Conference*

on Empirical Methods in Natural Language Processing, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 4945–4977. https://doi.org/10.18653/v1/2023.emnlp-main.302

Eli Schwartz, Leshem Choshen, Joseph Shtok, Sivan Doveh, Leonid Karlinsky, and Assaf Arbelle. 2024. NumeroLogic: Number Encoding for Enhanced LLMs' Numerical Reasoning. *arXiv preprint arXiv:2404.00459* (2024).

semantic release. 2024. *Semantic-Release*. Retrieved August 09, 2024 from https://github.com/semantic-release/semantic-release

Emad Shihab, Akinori Ihara, Yasutaka Kamei, Walid M Ibrahim, Masao Ohira, Bram Adams, Ahmed E Hassan, and Ken-ichi Matsumoto. 2013. Studying re-opened bugs in open source software. *Empirical Software Engineering* 18, 5 (2013), 1005–1042. https://doi.org/10.1007/s10664-012-9228-6

Edgar A Smith and RJ Senter. 1967. *Automated readability index*. Vol. 66. Aerospace Medical Research Laboratories, Aerospace Medical Division, Air ....

Davide Spadini. 2024. *PyDriller*. Retrieved August 26, 2024 from https://github.com/ishepard/pydriller

Martin Staffa. 2020. *angular.js/DEVELOPERS.md at master · angular/angular.js*. Retrieved August 13, 2024 from https://github.com/angular/angular.js/blob/master/DEVELOPERS.md#commits

Yingchen Tian, Yuxia Zhang, Klaas-Jan Stol, Lin Jiang, and Hui Liu. 2022. What makes a good commit message?. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 2389–2401. https://doi.org/10.1145/3510003.3510205

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 [cs.CL] https://arxiv.org/abs/2302.13971

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2024. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '22)*. Curran Associates Inc., Red Hook, NY, USA, Article 1800, 14 pages.

John Wieting, Taylor Berg-Kirkpatrick, Kevin Gimpel, and Graham Neubig. 2019. Beyond BLEU: Training Neural Machine Translation with Semantic Similarity. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Anna Korhonen, David Traum, and Lluís Màrquez (Eds.). Association for Computational Linguistics, Florence, Italy, 4344–4355. https://doi.org/10.18653/v1/P19-1427

Jianyu Wu, Hao He, Wenxin Xiao, Kai Gao, and Minghui Zhou. 2022. Demystifying Software Release Note Issues on GitHub. In *2022 IEEE/ACM 30th International Conference on Program Comprehension (ICPC)*. 602–613. https://doi.org/10.1145/3524610.3527919

Jianyu Wu, Weiwei Xu, Kai Gao, Jingyue Li, and Minghui Zhou. 2023. Characterize Software Release Notes of GitHub Projects: Structure, Writing Style, and Content. In *IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2023, Taipa, Macao, March 21-24, 2023*, Tao Zhang, Xin Xia, and Nicole Novielli (Eds.). IEEE, 473–484. https://doi.org/10.1109/SANER56733.2023.00051

XGBoost. 2022. *Understand your dataset with XGBoost*. Retrieved September 12, 2024 from https://xgboost.readthedocs.io/en/stable/R-package/discoverYourData.html#build-the-feature-importance-data-table

Wenxin Xiao, Hao He, Weiwei Xu, Xin Tan, Jinhao Dong, and Minghui Zhou. 2022. Recommending good first issues in github oss projects. In *Proceedings of the 44th International Conference on Software Engineering*. 1830–1842.

Zikai Xie. 2024. Order Matters in Hallucination: Reasoning Order as Benchmark and Reflexive Prompting for Large-Language-Models. *arXiv preprint arXiv:2408.05093* (2024).

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

Yuxia Zhang, Zhiqing Qiu, Klaas-Jan Stol, Wenhui Zhu, Jiaxin Zhu, Yingchen Tian, and Hui Liu. 2024. Automatic Commit Message Generation: A Critical Review and Directions for Future Work. *IEEE Transactions on Software Engineering* 50, 4 (2024), 816–835. https://doi.org/10.1109/TSE.2024.3364675

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2023. A Survey of Large Language Models. arXiv:2303.18223 [cs.CL] https://arxiv.org/abs/2303.18223

Zulip. 2024. *Comparing 9.0...9.1 · zulip/zulip*. Retrieved February 25, 2025 from https://github.com/zulip/zulip/compare/9.0...9.1

Received 2025-02-25; accepted 2025-04-01