Automated Grading and Feedback for Commit Messages Using NLP Models

Albert Fares, Daniel Polka, Hugo Jeannin

***Abstract*—This report presents an automated system for commit message grading and feedback generation, developed as part of the Machine Learning (CS-433) course project. Leveraging NLP techniques, we fine-tuned a BERT-based model to assign a grade, optimized its hyperparameters for improved performance, used GPT-4o for dynamic feedback generation, and implemented fine-grained error-checking techniques to identify specific areas of improvement. Rigorous data preprocessing, including dataset balancing and augmentation, proved critical to overcoming data limitations. The final system promotes adherence to best practices in commit message writing, providing clear, context-specific suggestions.**

I. Introduction

Commit messages play a crucial role in software development, serving as a bridge between developers and version control systems. Well-written commit messages improve collaboration, ease debugging, and provide a clear history of changes, while poor messages can hinder comprehension and reduce code maintainability. Despite their importance, writing high-quality commit messages remains a challenge, especially for students and novice developers.

In collaboration with the Dependable Systems Lab at EPFL, led by Professor George Candea, we developed an automated system to grade commit messages and provide constructive feedback. The system is designed for the SWENT (CS-311) course, where students write commit messages as part of their software development projects. The goal is twofold: (1) to assess the quality of commit messages based on established best practices, such as conventional commits, and (2) to generate actionable feedback that helps students improve their writing.

Our approach combines machine learning techniques with natural language processing (NLP). Specifically:

- A fine-tuned BERT model is used for commit message classification and grading.
- A GPT-4o model generates dynamic, context-specific feedback based on detected issues.

This report is organized as follows: Section II describes the datasets used and the preprocessing techniques applied to prepare the data. Section III explains the grading model, including model selection, fine-tuning, and error detection methods. Section IV presents the results, including model performance and examples of generated feedback. Finally, Section V discusses the challenges we faced, possible improvements and the impact of this system on improving commit message quality.

II. Methodology

A. Dataset

We used two datasets provided from the SWENT (CS-311) course:

- The first dataset contains **2,400 commit messages** labeled with grades (integers) ranging from 0 to 5. These grades reflect commit quality, adhering to either the *Conventional Commits* guidelines (max grade = 5) or the more lenient *SWENT style* (max grade = 4).
- The second dataset consists of graded commit messages with additional textual comments explaining the grades. These comments were manually written by teaching assistants.

We identified two key challenges with these datasets: (1) an imbalance in class distribution (overrepresentation of grades 3, 4, and 5) and (2) inconsistencies in the grades and justifications.

B. Preprocessing

To address dataset limitations and prepare the data for training, we implemented several preprocessing techniques. Below, we summarize both our exploratory methods and the final approach:

a) Exploratory Methods:

- **Oversampling:** To mitigate class imbalance, we oversampled underrepresented classes (like 0 and 1). This improved grade distribution but remained insufficient to augment and balance the dataset.
- **Synonym Replacement:** Augmentation techniques, such as random word insertion, swap, and deletion, were applied. However, due to the technical nature of commit messages, replacements often produced irrelevant or incorrect content (e.g., ”style” replaced by ”fashion”).
- **Synthetic Data Generation:** Using GPT-4o in a few-shot setting, we generated commit messages based on examples. While this approach expanded the dataset, many generated messages were repetitive and lacked diversity.

b) Final Approach: We adopted a simplified, context-focused grading system:

- Grades were mapped from the original **0–5 scale** to a **0–3 scale**, based only on the clarity and meaning of the commit description. Prefixes and body content are not given to the model for training or for evaluation.
- Table I shows the grade mapping:

TABLE I
Context Grade Mapping

Original Grade (/5)	Mapped Grade (/3)
5 & 4	3/3
3 & 2	2/3
1 & 0	1/3

To ensure consistency, we first removed duplicates where both the message and the grade were the same. We then performed

the mapping mentioned in table I and manually reviewed all messages with grade **1/3** after the mapping to correct some of them to **0/3**. Next, we removed each duplicate message by hand, having chosen the correct grade it should get ourselves. This gave us an initial distribution of 41 **0/3**, 64 **1/3**, 756 **2/3** and 1266 **3/3** messages.

Given this distribution, we decided to oversample class 0 by 2 and class 1 by 5. After the oversampling, we used GPT-4o to generate 100 new examples of extremely poor-quality messages (e.g., gibberish, single-word or off-topic) graded as **0/3**. Finally, we oversampled class 0 by 2 once more to end up with 364 **0/3**, 320 **1/3**, 756 **2/3** and 1266 **3/3** messages.

C. System Architecture

The system consists of two main components: a **grading model** for commit message evaluation and a **feedback generation model** for justification.

a) Grading Process: Our final approach combines a fine-tuned BERT model with rule-based post-processing:

- **BERT-Large-Uncased Fine-Tuning:** We fine-tuned the pre-trained `bert-large-uncased` model, accessible on Huggingface, using the final dataset. This model outputs an initial grade ranging from 0 to 3 based solely on the commit message's description.
- **Grade Mapping:** The initial grade (0–3) is mapped to a 0–5 scale, as shown in Table II.

TABLE II
FINAL GRADE MAPPING

Initial Grade (/3)	Final Grade (/5)
0/3	0/5
1/3	1/5
2/3	3/5
3/3	4/5

After the initial grade is predicted, we apply the following validations and adjustments:

- **Body Evaluation:** If a body is present, it is evaluated for:
  - **Meaningfulness:** The body must add value to the description.
  - **Line Length:** Each line must not exceed 72 characters.

  A bonus of **+0.5 points** is applied for each criterion met, based on the context grade.
- **Prefix Validation:** If a prefix is present, we apply the following bonuses:
  - **+1.0:** Prefix is valid and perfectly formatted.
  - **+0.5:** Prefix is valid but contains minor formatting issues.
  - **+0:** Prefix is invalid or not from the predefined list.
- **Error Penalties:** For all messages, we check for specific errors, each deducting **0.5 points**:
  - Description exceeding 50 characters (excluding prefix).
  - Absence of an imperative verb at the start of the description. This was validated using SpaCy to identify verbs in the imperative mood.
  - Capitalization of the first letter of the description (only for conventional commit messages).

b) Feedback Generation: Initially, we attempted to implement a fine-tuned version of GPT-4o by training it on one of our datasets, which contained commit messages and comments written by teaching assistants. However, the comments in this dataset were not detailed enough to effectively train the model, leading to subpar performance. For example, consider the following commit message, which is a perfect example of a SWENT-style message requiring no changes:

> *"Start location based toDos user story"*

Using the same structured prompt for both models, the fine-tuned GPT-4o produced the following feedback:

> *"The message is clear but could be more precise about the changes made."*

This feedback is vague and incorrect, failing to recognize the quality of the message or provide any actionable suggestions. In contrast, the non-fine-tuned GPT-4o, using the exact same structured prompt, produced far more actionable and accurate feedback:

> *"The commit message is clear and effective, following SWENT-style standards. While no improvements are necessary, you might consider adopting the conventional commit format for future alignment with widely used best practices."*

This shows that despite identical prompts, the fine-tuned model struggled due to its limited training data, often failing to provide accurate feedback. In contrast, the base GPT-4o, paired with structured prompts, delivered precise and actionable suggestions, demonstrating that the dataset was not good enough to justify fine-tuning with it and that qualitative prompting with the base GPT-4o model proved more effective.

The structured prompts play a critical role in ensuring the accuracy and relevance of the feedback. Each prompt is dynamically generated based on the specific errors or good practices identified in the commit message. This process involves using Boolean flags, which serve to indicate whether certain issues were detected and are used to construct a prompt that addresses only the relevant issues or simply acknowledges adherence to best practices.

To further enhance the quality of feedback, we set the GPT-4o model's temperature to 0.5, which introduces slight variability in responses to make them more human-like and conversational. The maximum output token limit was capped at 200, ensuring that the feedback remains concise yet sufficiently detailed to be actionable.

D. Training Details

The final `bert-large-uncased` model was fine-tuned using the following hyparameters:

- **Learning rate:** 2×10^{-5}
- **Batch size:** 16
- **Epochs:** 3
- **Weight Decay:** 0.0
- **Optimizer:** AdamW

The final training accuracy achieved was **0.79**.

III. FINAL WORKFLOW

The commit message grading workflow evaluates a commit message through a series of steps, combining model predictions and rule-based checks to assign a final grade. The workflow consists of:

- **Component Extraction:** The commit message is split into its components: prefix (optional), description and body (optional).
- **Initial Grading:** The description is graded using the fine-tuned BERT model, outputting an initial grade (integer) ranging from 0–3.
- **Prefix Validation:** For messages with prefixes, errors like formatting issues or invalid prefixes are detected and adjustments (bonuses or penalties) are applied.
- **Body Evaluation:** If a body is present, it is checked for line length with a function and for meaning using GPT-4o. If both criteria are satisfied, a bonus is applied.
- **Error Detection:** Specific rule-based checks (e.g., description too long, absence of imperative verbs) identify issues that result in penalties.
- **Dynamic Feedback Generation:** After detecting format errors, the full commit message along with its grade and the constructed prompt are passed as inputs to GPT-4o in order to generate feedback containing pertinent suggestions.
- **Final Adjustments:** The adjusted grade is mapped to a 0–5 scale and capped within valid bounds.

Note: See Figure 3 for visual representation.

IV. Results

A. Hyperparameter tuning

In order to choose the best hyperparameters for our model, we trained several models and chose the one with the best training accuracy. We fixed the number of epochs at 3 and varied the batch size (16, 32), learning rate (2×10^{-5}, 3×10^{-5}, 5×10^{-5}) and weight decay (0.0, 0.05, 0.1). Figure 1 shows the training accuracy results of all possible combinations of these values, the best one being the one we chose as final model.

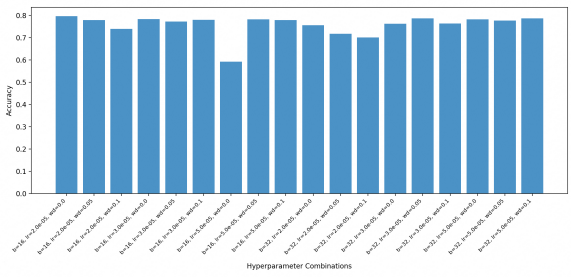

Fig. 1. Hyperparameter combinations and respective training accuracies

B. Dataset comparison

After concluding that the original dataset was sub-optimal (the one without feedback), we decided to enhance it as described in section II.

Table III and figure 2 summarize these performance metrics, with the original dataset's messages (containing prefixes and bodies) being graded from 0-5, and the final dataset's messages (without prefixes or bodies) being graded from 0-3. To compare these datasets, we used them to train the `bert-large-uncased` model with the following hyperparameters: 3 epochs, batch size = 16, learning rate = 2×10^{-5}, weight decay = 0. To evaluate their performance, we used the dataset with feedback (using only the grade as label).

TABLE III
Performance Comparison of Grading Models

Metric	Original dataset	Final dataset
Accuracy	0.43	0.53
Precision	0.50	0.65
Recall	0.43	0.53
F1 Score (Weighted)	0.45	0.56

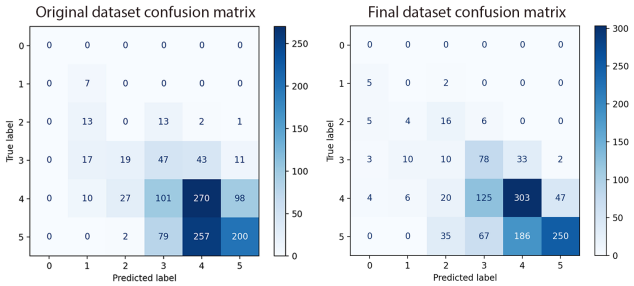

Fig. 2. Confusion matrices for old and new datasets

C. GPT-4o Feedback Outputs

Table IV showcases examples of feedback generated by GPT-4o for actual commit messages written by students. These outputs illustrate how the model evaluates the quality of the messages, identifies specific issues, and offers actionable suggestions to enhance clarity and adherence to best practices. The examples also demonstrate the model's ability to provide precise feedback, ranging from identifying typographical errors to suggesting structured and detailed improvements.

TABLE IV
Examples of Commit Messages, Grades, and Feedback

Grade	Example Message	GPT-4 Justification
0	"Faet: Updated Deliverables.md"	The commit message lacks important details about the changes made to Deliverables.md, making it insufficiently specific and descriptive. The description should start with a lowercase letter and use an imperative verb to clearly state the intent, aligning with conventional commit message practices. The prefix contains uppercase letters and a typo; it should be lowercase and correctly spelled, such as 'feat'.
5	"test: add tests for EditToDoScreen. \n\nWe check wether the view model get updated correctly."	The commit message contains a spelling mistake: "wether" should be "whether." Additionally, while it mentions that the view model gets updated, it could provide more detail about what specific aspects or functionalities are being tested. Otherwise, it adheres well to conventional commit standards.

V. Discussion

Our work demonstrates the potential of combining a fine-tuned BERT model for automated grading with GPT-4o for dynamic feedback generation, offering a robust solution for commit message evaluation. This approach enhances developer workflows by automating assessment and providing actionable guidance, making it particularly valuable for educational purposes or quality enforcement in software development projects.

A. Strengths of the Approach

- **Dynamic Feedback Generation:** GPT-4o generates targeted, human-friendly feedback from the detected errors. This helps to pinpoint specific issues and educates

developers on best practices, encouraging continuous improvement in commit message quality.
- **Adaptability:** The modular design allows for easy adjustments to grading rules or prefix conventions, making the system flexible for other projects and development standards.

B. Challenges and Limitations

Despite the system's strengths, several challenges were encountered during development:

- **Data Imbalance:** The original dataset had a skewed distribution, with an overrepresentation of high-quality commit messages (grades 3–5). While oversampling and synthetic data generation mitigated this issue, further improvements are needed to ensure better generalization.
- **Inconsistent Human Labels:** Grading inconsistencies in the dataset impacted model performance. Manual review and cleaning helped resolve some discrepancies, but more robust data validation techniques could further improve reliability.
- **Computational Overhead:** Fine-tuning the BERT model required significant computational resources, which may limit scalability. Training on EPFL's Izar cluster helped with this, but more efficient solutions could be explored.

C. Ethical Risks

While the automated grading system provides significant benefits in terms of efficiency and consistency, several ethical risks should be considered:

- **Bias in Model Predictions:** The model's performance is influenced by the quality of the training data. If the data used for training contains biases (e.g., skewed towards certain commit styles or types of errors), the model could perpetuate these biases. This could lead to unfair grading for certain types and/or formulations of commit messages.
- **Overreliance on Automation:** While automation can improve efficiency, overreliance on the system for grading and feedback may limit developers' ability to critically evaluate their own work. It is essential to maintain human oversight to ensure that the system does not undermine critical thinking or creativity.
- **Privacy Concerns:** Since commit messages may contain sensitive information about the code or project, there are privacy concerns related to the data used for training and the feedback generated. It is necessary to ensure that this information is handled responsibly and anonymized where necessary to properly protect users' privacy.

D. Future Improvements

To further enhance the system, several avenues for improvement can be explored:

- **Sentiment Analysis:** Integrating sentiment analysis into the grading process could help evaluate message tone and clarity, particularly for vague or overly brief descriptions.
- **Real-Time Integration:** Deploying the system as a real-time tool (e.g., Git hooks or IDE plugins) would allow developers to receive immediate feedback during the commit process, enhancing usability and impact.
- **Retraining the Model:** If the system is adopted for the SWENT course, the model could be retrained on future double-checked versions of its predictions. This iterative improvement would help refine accuracy and better align the system with course requirements.

VI. Conclusion

This paper demonstrates the effectiveness of using NLP techniques to automate commit message evaluation and feedback generation. By combining a fine-tuned BERT model for grading with GPT-4o for dynamic feedback, we developed a robust system that streamlines commit message assessment and promotes adherence to best practices.

The fine-tuned `bert-large-uncased` model delivers reliable and consistent grading of the quality of a commit's description, while rule-based validations ensure alignment with guidelines such as Conventional Commits. GPT-4o complements this grading process by generating clear, actionable justifications based on detected errors, helping developers understand and improve their commit messages.

Key achievements of this work include:

- Developing a flexible grading system that evaluates commit messages based on context, structure, and adherence to conventions.
- Addressing dataset limitations through preprocessing, class balancing, and manual validation, ensuring robust model performance.
- Creating a dynamic feedback mechanism that educates developers and provides targeted suggestions for improvement.

Despite its success, the system faces challenges such as data imbalance, computational costs, and limited diversity in synthetic data generation. Addressing these issues would be crucial for further enhancing its scalability and performance.

Looking ahead, the integration of sentiment analysis, domain-specific pre-trained models and real-time deployment as Git hooks or IDE plugins can significantly extend the system's applicability and greatly refine the system's grading accuracy.

In conclusion, this project highlights the potential of automated grading systems to bridge the gap between manual code review and NLP-driven solutions. By improving commit message quality, our system enhances collaboration, version control history and educational outcomes, making it a valuable tool for both software engineering courses and professional development workflows.

APPENDIX

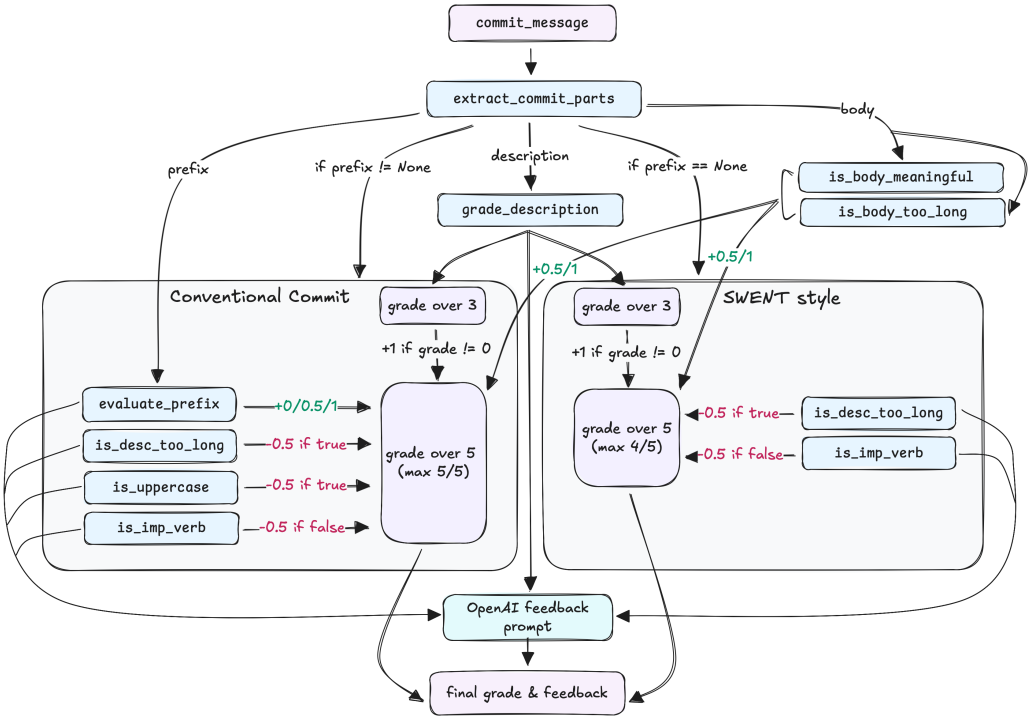

Fig. 3. Commit Message Grading Workflow