


Fine-Tuning Codet5 for Commit Message Generation from Source Code Diffs

Jasmina Vulovic 

Informatics: Games Engineering , Technical University of Munich

 jasmina.vulovic@tum.de

April 7, 2025

1 Introduction

This project involves fine-tuning a large model on a synthetic dataset designed to simulate realistic commit patterns across various technologies, specifically focusing on generating commit messages based on code diffs. The synthetic approach was chosen to meet the criteria of the JetBrains internship application, ensuring flexibility across different commit scenarios. We then train a sequence-to-sequence transformer model (CodeT5[5]) to generate commit messages from simulated code changes.

2 Related Work

In recent years, several studies have focused on automating the generation of commit messages from code changes. Jung (2020)[3] introduces a method for generating commit messages using a 345K dataset and CodeBERT, marking the first multi-language fine-tuning approach for this task. Building upon this, RACE [6] was introduced, a retrieval-augmented method for commit message generation, using similar commits as exemplars. Authors' approach, tested on five languages, outperforms existing models and improves Seq2Seq performance.

3 Methods

3.1 Data creation

The synthetic data was generated by balancing structured patterns with realistic commit message types: `feat`, `fix`, `refactor`, `docs`, `test`, `perf`, `chore`, `build`, and `ci`. Each follows the conventional commit format with placeholders for components. The vocabulary includes 26 software components (e.g., *API*, *database*), 20 development actions (e.g., *add*, *remove*), and 21 feature categories (e.g., *user authentication*). Specialized vocabularies are used for specific commit types, such as `fix` descriptions. The generation process randomly samples templates and vocabulary,

adding variations like version tags and random words for uniqueness.

Data-set Results: 1,500 total samples, out of those 1,200 are training samples (80%) and 300 validation samples (20%). We have 99% unique commit messages.

3.2 Preprocessing

The raw synthetic data is processed by converting the format to "Repo: repo, Diff: diff" and tokenizing it using the CodeT5 tokenizer with a max length of 256 for inputs and 36 for commit messages. Padding tokens are replaced with -100 for loss calculation. The dataset is then prepared for training by creating torch-compatible sets, removing original columns, and applying dynamic padding, before splitting it into training and validation sets.

4 Training Setup and Custom Metric Logging

This section outlines the training configuration, enhanced metric calculation, and custom callback implementation for logging during model training.

4.1 Training Setup

The model is fine-tuned using the following settings:

The model used a batch size of 8 for training and evaluation, with a learning rate of 2×10^{-5} and weight decay. Training ran for 10 epochs, evaluating and saving the model every 100 steps. Gradients were accumulated over 2 steps for larger batch sizes, and the best model was selected based on the ROUGE-L[2] score.

4.2 Metric Calculation

During training, several metrics are computed to evaluate the quality of the generated commit messages: The BLEU score measures the overlap between n-grams

in the predicted and reference texts, while the METEOR[4] score evaluates translations based on precision, recall, synonymy, and word order. The ROUGE-2 score assesses bigram overlap between the generated and reference text, indicating diversity, and Prefix Accuracy tracks how often the first words of the generated text match the reference.

4.3 Custom Callback for Metrics Logging

The `PrintMetricsCallback` logs key metrics during training, including training loss, BLEU, METEOR, ROUGE-2, and Prefix Accuracy. It also records the learning rate and evaluation metrics at each step, if available. Data is logged only by the local process to ensure efficient training. Metrics are displayed clearly, separated by "=", providing valuable feedback to track progress and make adjustments.

5 Results

5.1 Training Loss and Evaluation Results Interpretation

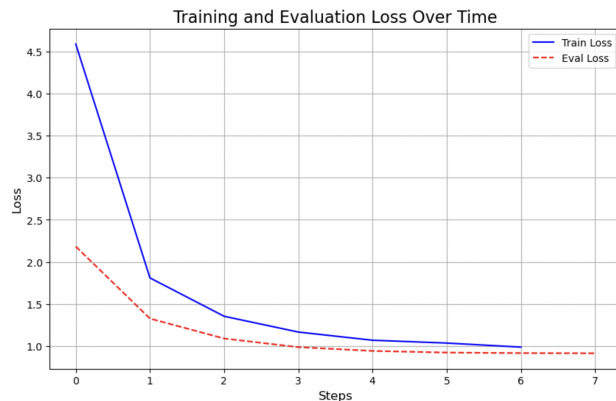


Figure 1 Loss Results during Training and Evaluation.

The training loss decreases steadily over time, starting at around 4.59 at step 100 and gradually reducing to about 0.91 by step 750, indicating that the model is learning well. The loss reduction is smooth, showing that the model is converging without any sharp changes, which is a good sign of stable training.

The evaluation loss follows a similar pattern, decreasing from 2.18 at step 100 to 0.91 at step 750. The evaluation loss is consistently lower than the training loss, and the gap between them narrows over time. This suggests that the model is generalizing well on unseen data, not overfitting.

Overall, the model is performing well, showing steady progress in both training and evaluation metrics. It's learning and generalizing effectively, with no obvious signs of poor performance or overfitting. However, since the model has been trained on a small synthetic dataset, this is its main limitation.

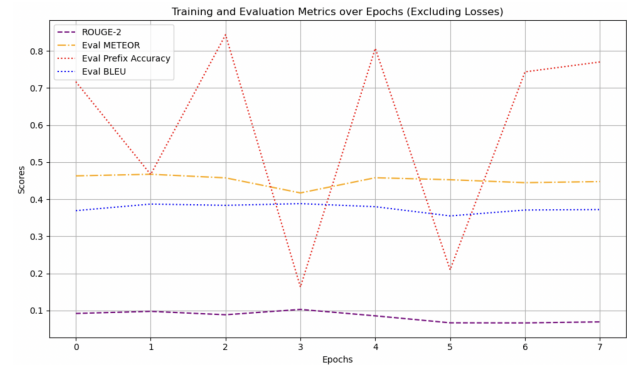


Figure 2 Interpreting the Metrics Results

5.2 4.2 Interpreting Metrics Results

The BLEU[1] score (0.35–0.39) shows the model is learning to generate text that aligns with the reference, though there's still room for improvement. The METEOR score (0.42–0.47) indicates solid performance in producing semantically meaningful text, with consistent results pointing to growing language understanding. ROUGE-2 (0.066–0.102) reflects gradual improvement in capturing bigram overlaps, suggesting better word pairing with more training. Exact-Prefix-Matching struggles but stabilizes towards the end, showing progress despite challenges.

The model's performance is solid despite the limitations of synthetic data, with no overfitting and steady loss decrease, indicating good generalization and fine-tuning. Better data would improve these results further.

6 Conclusion

In conclusion, this project shows that fine-tuning the CodeT5 model for generating commit messages from code diffs can be effective. By using a synthetic dataset with different commit types and vocabularies, we trained a model to produce relevant commit messages. The evaluation results, such as BLEU, METEOR, and ROUGE-2 scores, show good progress, with the model improving over time. While the synthetic data has limitations, the model performs well and can be improved further with more diverse data.

References

- [1] *BLEU - a Hugging Face Space by evaluate-metric*. URL: <https://huggingface.co/spaces/evaluate-metric/bleu>.
- [2] PyTorch-Ignite Contributors. *RougeL — PyTorch-Ignite V0.5.2 documentation*. URL: <https://pytorch.org/ignite/generated/ignite.metrics.RougeL.html>.
- [3] Tae-Hwan Jung. *CommitBERT: Commit Message Generation using Pre-Trained Programming Language Model*. May 2021. URL: <https://arxiv.org/abs/2105.14242>.
- [4] Alon Lavie and Abhaya Agarwal. “METEOR: An automatic metric for MT evaluation with high levels of correlation with human judgments”. In: (July 2007), pp. 228–231.
- [5] Salesforce. *codet5-base*. <https://huggingface.co/Salesforce/codet5-base>. Accessed: 2024-01-18. 2024.
- [6] Ensheng Shi et al. “RACE: Retrieval-Augmented Commit Message Generation”. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing* (Jan. 2022), pp. 5520–5530. DOI: 10.18653/v1/2022.emnlp-main.372. URL: https://aclanthology.org/2022.emnlp-main.372/?utm_source=chatgpt.com.