Fine-Tuning CodeT5 for Predicting if-Statements in Python

1. Introduction

This project involved the fine-tuning of the pre-trained transformer model CodeT5 (specifically Salesforce/codet5-small) for the task of masked code completion, focusing on predicting missing if conditions in Python functions. The model was trained to infer the correct logical condition when provided with a function where the if condition was masked.

The task is significant for code intelligence applications such as automated code generation, refactoring, and repair. The small variant of CodeT5 was used due to its manageable parameter size (approximately 60 million) and balance between efficiency and capability.

2. Dataset Preparation

The dataset provided consisted of:

- 50,000 training samples
- 5,000 validation samples
- 5,000 test samples

Each record in the dataset included:

- cleaned_method: a flattened version of a Python function with an if condition
- target_block: the corresponding if condition to be predicted

Preprocessing Steps:

- The if condition within each function was replaced with a <mask> token.
- Code was flattened into a single-line representation.
- Tokenization was performed using the AutoTokenizer associated with the CodeT5 model.

Example: **Original:**

```
def check_positive(num):
    if num > 0:
        return "Positive"
    else:
        return "Non-Positive"
```

Masked Input:

def check_positive(num): <mask>: return "Positive" else: return "Non-Positive"

Target: if num > 0

---

3. Fine-Tuning Process

The model was fine-tuned using the Hugging Face Trainer API with the following configuration:

- Model: Salesforce/codet5-small
- Optimizer: AdamW
- Learning rate: 3e-4
- Epochs: 5
- Batch size: 4 (with gradient accumulation for memory efficiency)
- Early stopping: Enabled (based on validation loss)
- Precision: Mixed-precision training (FP16)

Fine-tuning was conducted on Google Colab using a T4 GPU. Checkpoints were saved at each epoch, with the best-performing model (lowest validation loss) retained.

---

4. Evaluation

The evaluation was conducted on the 5,000-sample test set. The following metrics were used to assess model performance:

- **Exact Match:** Measures the percentage of predicted conditions that exactly match the reference.
- **BLEU-4:** A standard machine translation metric that computes n-gram overlap up to 4-grams.
- **CodeBLEU:** A code-specific metric considering syntax, structure, and data flow, using a simplified token overlap version for fast evaluation.

Each prediction was compared with its ground truth, and the results were compiled into testset-results.csv with the following fields:

Input function with masked if condition, Expected if condition, Predicted condition, Exact match (true/false), BLEU-4 score, CodeBLEU score

---

5. Results

Model performance on the test set was as follows:

Metric	Score (Approximate)
Exact Match	0.0%
BLEU-4	0.42
CodeBLEU	~0.41

The exact match score indicates that the model rarely produced condition statements that were identical to the target output. This was expected, given the subtle variations possible in logical expressions and the relatively small training subset used due to hardware constraints. It is important to note that I ran into issues while trying to use Colab, as there was limited RAM space and my code was not able to successfully compile 50,000 units without crashing.

However, the BLEU-4 and CodeBLEU scores, while low, suggest that the model was still able to generate condition expressions that were structurally and semantically related to the correct answer. For example, the model often generated partial matches such as using the correct variable or comparison operator but failing to reproduce the full expression exactly.

The similar values of BLEU and CodeBLEU suggest that the model effectively captured both surface-level token patterns and some deeper structural elements of conditional logic. These results indicate that further fine-tuning, **especially** with a larger dataset or extended training time, could enhance the model's ability to generalize and produce accurate conditional statements. Additionally, employing techniques such as data augmentation, curriculum learning, or scaling up to the full CodeT5 model may improve performance on this type of detailed code synthesis task.

---

6. Conclusion

This project focused on fine-tuning the codet5-small model to predict masked if conditions within Python functions. The goal was to train the model to generate accurate logical conditions given the structure and content of a function. The process involved modifying a provided dataset of 50,000 functions by masking the conditional logic, flattening the code, and preparing it for sequence-to-sequence training using Hugging Face’s Trainer API.

While resource limitations made it difficult to complete training on the full dataset at once, the model was successfully fine-tuned on a reduced subset and evaluated on 300 test samples. This produced measurable results, including BLEU and CodeBLEU scores, and helped demonstrate the model’s ability to learn structural patterns in code. The project also showed the importance of preprocessing, memory management, and checkpointing when working with large models in constrained environments.

GitHub repository: https://github.com/amoolyat/training_CodeT5