Fine-Tuning Process

The fine-tuning process for CodeT5-small involved preparing a dataset of Python methods that contained if statements, with the objective of training a model to correctly predict the masked if condition. The model selected was Salesforce/codet5-small, which is a pre-trained language model suitable for code-related tasks. The fine-tuning procedure was conducted using the HuggingFace Trainer API, with early stopping based on evaluation loss in order to prevent overfitting.

Training was done over seven epochs, with checkpoints saved at each epoch to allow for resuming training if needed. The learning rate was set to 3e-4 with a batch size of 8 for both training and validation. Input sequences were truncated to a maximum of 512 tokens, and target sequences to 64 tokens, in order to ensure compatibility with the CodeT5 tokenizer and memory constraints. The final model that was used was the one with the lowest evaluation loss across all 7 epochs.

Dataset Preparation

The training, validation, and test datasets were loaded from CSV files, each containing Python methods along with their corresponding if-condition blocks. For each example, the target if condition was masked out and replaced with "<mask>" in the full method body, forming the input to the model. The target for prediction was the removed if-condition block.

During preprocessing, special characters in the if condition were handled by adding escape characters, allowing accurate and safe replacements in the code.The masked methods were flattened by removing excessive whitespace, newlines, and tabs, which allowed for a consistent input format for the model. A custom PyTorch Dataset class was used to tokenize and wrap the inputs and targets into a format the model could read.

Evaluation Metrics and Results

The evaluation of the model was conducted using several metrics including Exact Match, BLEU-4, CodeBLEU, and F1 Score. The BLEU-4 score was computed using the SacreBLEU library, and the exact match was determined by a direct comparison of the predicted and reference if conditions. The model achieved a BLEU-4 score of 100.0 and an exact match accuracy of 96.2%, indicating strong similarity of syntax between predictions and references.

The F1 score, which balances both precision and recall, was also calculated and resulted in 78.82%. This metric provides a more nuanced view of the model's accuracy, particularly in cases where predictions may not be exact matches but still capture the correct meaning.The F1 score highlighted the model's capability to capture relevant patterns even when the exact token structure differed.

Challenges with CodeBLEU

While BLEU and F1 scores were successfully computed, evaluating the model using CodeBLEU presented multiple challenges. Initially, the CodeBLEU repository by Salesforce could not be cloned due to access errors. So, as an alternative, the Microsoft CodeXGLUE version was used. However, this introduced another layer of problems due to deeply nested directories and missing build.sh scripts, which are essential for building the language parser required by CodeBLEU.

Even after resolving the directory structure and successfully building the tree-sitter parser for Python, the runtime environment in Google Colab failed to locate the required calc_codebleu.py script. Manual attempts to run CodeBLEU through subprocess calls returned non-zero exit statuses, and the expected output was not generated.

To work around these issues, I attempted to locate and run calc_codebleu.py independently, but the Salesforce-hosted script had been removed from GitHub, resulting in a 404 error. At this point, all potential sources for running CodeBLEU either failed due to missing files or unmet dependencies. Further debugging indicated that even when the script was added manually, required components like tree_sitter.Language initialization failed due to incorrect usage or missing parameters.

Next Steps for Tuning CodeBLEU

To enable CodeBLEU evaluation, several tuning steps are necessary. First, a stable and complete version of the CodeBLEU evaluation script must be downloaded or reconstructed. This includes ensuring that the calc_codebleu.py file and all parser dependencies are accessible and correctly built in the expected directory.

Second, the Colab environment must be consistently pointed to the correct working directory using %cd commands, and the build process for tree-sitter languages should be verified using ls to confirm the presence of build.sh. Lastly, to avoid runtime import errors, it's essential to test each Python module manually before running the full evaluation script.

Until these issues are resolved, the CodeBLEU scores will default to 0.0, which does not accurately reflect the model’s performance.