Summary of Paper

- Prior work on generative models for code focuses on creating new code,ignoring unique requirements for editing existing code.
- Author explore multi-round code auto-editing,it predicts code region based on recent changes.
- Coeditor outperform best code completion approach,double match accuracy,being a smaller model it demonstrate advantage of inclusion of editing history.
- Goal here is to predict edits to code conditioned on the user previous edits.

  $$P(\Delta u \mid \Delta_k \dots \Delta_1, U) .$$

  Here delta u represent region to modify in code U and delta1, …. deltak represents previous modification
- To solve this author proposed model CodeT5.It is based on two ideas:
  - It encodes all prior code edits using a line based diffing scheme and decodes using masked span filling
  - It uses lightweight static analysis to pull in relevant parts of codebase U.
- Challenges faced by author and solution
  - Handling of large context with numerous code changes : It uses block sparse attention instead of CodeT5 dense attention,which reduces the computation cost as well as maintain ability to all relevant code changes [flash attention].
  - Lack of suitable training data for multi-round auto-editing : Collected new dataset PYcommits (open source python project on github)
- Methodology
  - Encoding Code Changes
    - Authors want to select format that encodes and decodes code changes in a uniform manner while minimizing the number of tokens
    - Adopted line diff-based format to convert auto-editing to masked span infilling problem
    - Block of code made of lines l1,l2..lm user specified spans between a to a+n.Variable si indicates type of change (empty,add,delete).Encoding is done using below function

      $$\text{EncInput}(u) = s_1 l_1 s_2 l_2 \dots \texttt{<1>} s_a l_a \texttt{<2>} s_{a+1} l_{a+1} \dots \texttt{<n>} s_{a+n} l_{a+n} \dots s_m l_m .$$

    - Contextual changes are encoded with same format but with an empty edit region
    - Encoding of target region is done in format shown below

      $$\text{EncOutput}(\Delta u) = \texttt{<1>} I_a D_a \texttt{<2>} I_{a+1} D_{a+1} \ \dots \ \texttt{<n>} I_{a+n} D_{a+n} ,$$
      $$\text{where } I_i = \texttt{<add>} \ l'_{i1} \texttt{<add>} \ l'_{i2} \ \dots \ \texttt{<add>} \ l'_{i|I_i|} ,$$
      $$D_i = \text{if } l_i \textit{ is to be deleted} \text{ then } \texttt{<del>} \text{ else } \texttt{(empty)} .$$

  - Analyzing relevant signatures

        - Lightweight static analysis
            - Author analyse its pre-edit code and generate list of usages [For e.g. we retrieve its function signature]
            - Concatenate all these usage into a single document
            - Allow model to access the most important information about the current code region and improves performance significantly while generating only a small number of tokens

    - Adapting CodeT5 arch
        - Author arch was pre-trained and arch was based on CodeT5
        - Trained on large corpus of code using masked span infilling objective
        - Used CodeT5-base model [220M params] fine-tune it for their code auto-editing setting.
- PyCOMMITS dataset
    - Gather real-world code changes from commit history of open-source Python projects
    - Identify changes in code and separate commit into a list of unit (additions , deletions and modifications)
    - Only unit modifications are used as training labels while other two types of changes remain visible to the model as context.
    - Author created a training problem instance that instruct the model to predict the code changes based on all prior changes from the same commit
    - They employ a simple heuristic that sorts unit changes according to their source code locations and the import order between modules as git does not record editing order.
    - For proposed multi-round editing setting ,they generate synthetic data demonstrating repeated editing to the same code.
- Training Setup
    - Start lr - 2e-5
    - Optimizer - AdamW
    - Weight_decay - 1e-2
    - Scheduler - linear lr scheduler
    - Training steps - 1.34 million
    - Batch size - 1
- Results

Table 3: Performance on 5000 code completion instances extracted from edits (PyCommits-OneLine). Add EM and Replace EM are the (enhanced) exact-match accuracies on addition and replacement change, respectively.

Model	Parameters	Add EM (%)	Replace EM (%)	Overall EM (%)
InCoder1B	1.3B	29.0	25.2	26.2
InCoder6B	6.7B	34.0	30.4	31.3
SantaCoder	1.1B	31.0	28.1	28.8
Coeditor	220M	**47.1**	**64.9**	**60.4**

    - Enhanced exact-match (EM) accuracy metric that performs semantic-preserving code normalization before checking string equivalence
    - Results are measured on 5000 code completion problem sampled from there test dataset.Coeditor outperforms other model in both addition and replacement changes

- To evaluate Multi-round editing
  - To evaluate workflow they used ground truth changes to simulate the user's actions.Model predicts list of changes they compare with the ground truth if it exactly matches.If none of the suggested changes match the ground truth, they assume the user will manually perform the first remaining change
  - Metric they used
    - Lines - simply measures the number of changed lines before and after edit
    - Levenshtein - It measures number of character addition , deletion,and substitution needed to transform one string to another

Table 4: Multi-round evaluation results measured on 5000 problems from the PyCommits test set. Lines, Levenshtein, and Keystrokes are the average total gains in the corresponding metrics. Rounds is the average number of rounds needed to complete all desired changes.

Setting	Lines (%)	Levenshtein (%)	Keystrokes (%)	Rounds
SingleRound	28.5	23.1	19.2	1
MultiRound	46.7	25.9	28.6	2.43

Shortcomings of this work

- Not trained on larger sequence length by taking the advantage of dilated attention / flash attention which would help in generalisation and improvement of performance.
- Manual Identification of region of code requiring changes and then ask model to predict where and how to make these changes within region
- There model is designed not to modify any lines that have been already modified
- User cannot also partially edit a line and let model complete the rest of that line

Generalisation of this work

- First of all we can improve model performance by training the architecture on larger sequence length (as recent llm's are being trained on larger sequence length and have show significant improvement) with the help dilated attention [Link] or Sparse Flash attention [Link]
- We can replace Github Copilot with the above models as it takes advantage by tracking programmer changes and performance is better than it
- Also introduce this in IDEs (Like VS code,etc.) as most of the coder are IDEs for development
- We can use PEFT methods to finetune our model on new dataset (on future commits) as it saves lot of computation and training time.