ASSIGNMENT 1 REPORT

1 DATASET CONSTRUCTION

The objective of data creation was to build a high quality corpus from available Python projects. The created corpus would then support pre-training a transformer model using some pre-defined training objective (MLM, CLM, etc). The corpus would further be used to finetune a specific task: recommending if conditions.

1.1 REPOSITORY SOURCING AND RATIONALE

GitHub was used as the major source for Python projects for this assignment. The Github repositories were queried based on star count and the inclusion of Python (.py) files within the repository. The forks of the Github repositories were excluded to reduce duplication. Python Notebooks (.ipynb) were excluded as they have different source formatting than python files and subsequently degrade the tokenizer quality. Several domains including data tools, web backends, machine learning scripts, etc were queried to improve generalization and to extend the breadth of the corpus.

1.2 LOCAL CLONING AND FILE FILTERING

Each selected repository was first cloned to the local device. The files inside each repository were recursively listed and only Python sources were retained. Common files such as readmes, tests, build scripts, configurations and auto-generated files were removed. For every kept file the repository name and latest commit SHA were recorded to enable tracing and deduplication.

1.3 FUNCTION EXTRACTION

Python's `ast` was used to identify `FunctionDef` nodes and slice out exact code excerpt using line numbers.Very short functions (< 3 lines) and very long functions (> 200 lines) were defined as outliers and not included in the corpus. For each function metadata consisting of function name, length, number of if statements and cumulative length of if statements were stored for further corpus analysis.

1.4 QUALITY CONTROL

Unparsable files, duplicate functions and docstring functions were discarded. Functions with excessively long if targets and functions with empty if conditions were also discarded. These steps were taken to reduce the noise in the training data.

1.5 DATASET

The functions extracted after following these steps were stored in a .json file with metadata specifying their parent repository, last commit SHA, function name, function length, number of ifs in the function and the total cumulative length of the if statements. A brief analysis of the created corpus has been presented below:

2 TRAINING PROCESS

2.1 TOKENIZER TRAINING

All accepted function bodies constitute the pre-training text corpus used to train a byte-level BPE tokenizer from scratch (vocabulary ($32, \mathrm{k}$)). Training a tokenizer on in-domain code ensures stable subword segmentation for identifiers, punctuation, and operators, avoiding leakage from external pretrained tokenizers. The tokenizer was configured with a ByteLevel pre-tokenizer and decoder to preserve exact byte boundaries and reversible whitespace. This eliminated random characters and avoided inserting or removing spaces around punctuation when decoding model outputs. The trained tokenizer is reused consistently across pretraining, fine-tuning, and evaluation.

Metric	Value
Repositories (unique)	491
Functions parsed (total)	1,197,025
Avg. function length (lines)	18.61
Median function length (lines)	9
% functions with at least one `if`	41.07%
% functions with nested `if` (#ifs $>$ 1)	23.20%
Avg. cumulative `if` body length (lines)	5.84

Table 1: Corpus statistics computed from collected python functions.

2.2 Pretraining the Model

A Transformer was pre-trained on the function corpus with a Masked Language Modeling (MLM) objective 15% masking. The goal was to learn robust code token distributions and infilling behavior that transfer to code generation. For each function in the dataset, approximately 15% of the tokens were replaced with the masks described in the tokenizer i.e `<mask>`. The inputs were randomly masked and labels equal the original tokens only at masked positions.

From the serialized corpus, each function was filtered to functions within line-length bounds and normalizing whitespace. The corpus was split into train and validation sets using 95% for training and 5% for validation, resulting in $\approx 142{,}500$ training instances and $\approx 7{,}500$ validation instances for $150{,}000$ pretraining rows. Sequences were truncated to a maximum length of 256 tokens for efficient memory use.

The T5 model was pretrained with the Hugging Face Trainer. Representative settings:

1. per-device batch size: 16
2. AdamW with learning rate: 5×10^{-4}
3. weight decay: 0.01
4. `bf16`: True
5. gradient_checkpointing: True

2.3 Fine-tuning the model

The downstream task was to predict a masked single if statement inside a Python function. For each function, exactly one if condition was masked and the original condition was used as the target text. AST-based extraction was used to recover the source slice of the condition to avoid regex mismatches. The resulting masked-original pairs were split into train/validation/test sets (80/10/10). To reduce leakage from pretraining, fine-tune candidates were drawn from a disjoint pool.

Two label formats were considered for fine-tuning:

1. **Span infilling:** input is the masked function and the decoder generates the full condition, with variable length.
2. **Token-per-mask:** Early stopping was enabled to only predict the exact number of tokens as in the target.

Evaluation Protocol At evaluation time, each external function is masked in the first if test via AST and fed to the model to generate a condition. Two metrics were computed: (i) **Exact Match (EM)** on whitespace-normalized strings and (ii) **token-level F1** using simple whitespace tokenization. The reported scalar score is $(100 \times \max(\mathrm{EM}, \mathrm{F1}))$, so exact matches achieve 100, while partial overlaps receive graded credit.

Results The evaluation was done on two splits: an internal test set sampled from the same construction pipeline (generated-testset.csv) and an external provided test set (provided-testset.csv). Exact Match (EM), token-level F1, and the composite score (100×max(EM,F1)) have been reported in the table.

Split	#Examples	EM (%)	F1 (%)	Avg. score
Internal	4454	3.34	17.48	25.66
External	5000	2.80	15.92	23.41

Table 2: Evaluation on internal and external test sets.