PyLinguist: Automated Translation of Python for Hindi Programmers

First Author
Antara Tewary
G01413546
atewary@gmu.edu

Second Author
Ankit Kumar
G01436204
akumar37@gmu.edu

Third Author
Homa Haghighi
G01436204
akumar37@gmu.edu

1 Introduction

Python is a popular programming language, which has gained its popularity due to many factors, which include its readability and intuitive English-like syntax that makes the code accessible to English speakers. However, this can be a significant barrier for non-English speakers who have to grapple with both the programming concepts and learning a new language. Our project addresses this issue by developing a comprehensive translation system that converts Python code from English to Hindi, making Python programming more accessible to Hindi speakers while maintaining code functionality and readability.

1.1 Task / Research Question Description

Core research questions are-
- How can we effectively translate Python's syntax, keywords, and natural language elements while preserving code functionality?
- What combination of translation techniques provides optimal results for code translation??
- How can we quantitatively and qualitatively evaluate the effectiveness of code translation?

Our project implements a three-stage translation pipeline combining keyword dictionaries, Google Translate API, and GPT models, with comprehensive evaluation metrics to assess translation quality and code functionality preservation.

1.2 Motivation & Limitations of existing work

While Python's pseudo-code nature simplifies programming for English speakers, it presents significant challenges for non-English speakers who comprise the majority of the global population. Research has shown that students learn programming concepts more effectively when using a coding language based on their native language. Current solutions like CodeInternational partially address this issue by translating comments and identifiers, but they fall short of providing a complete solution that encompasses Python's built-in functions, keywords, and error messages.

Limitations of existing work include:

- Incomplete coverage of Python's language elements
- Lack of consistency in technical term translation
- Limited handling of language-specific challenges
- Insufficient preservation of code functionality
- Absence of comprehensive evaluation metrics

1.3 Proposed Approach

Our solution implements a three-stage pipeline that combines rule-based translation, neural machine translation and large language model enhancements:

- **Stage 0: Data Preprocessing**
  Initial stage aimed to prepare high-quality Python code data for training and evaluation purposes. We used the "python-code-dataset-500k" dataset from hugging face (jtatman, 2024) which provides a diverse collection of Python code exmaples. This stage involves-

  - Downloading and cleaning the dataset
  - Extracting Python code that was enclosed in the `<pythoncode>` tags using regex
  - Storing the cleaned code in a structured CSV format with `English_code` column.

- **Stage 1: Initial Translation**
  The first translation stage combines keyword mapping with neural machine translation:
  - Used curated keyword dictionary created by Joshua Otten (Otten et al., 2021) which 223 keywords in python and their Hindi translations.
  - Direct translation of keywords using this dictionary
  - Then, we apply Google Translate API for translating the comments, strings and natural language elements.
  - Special handling of compound variables with underscores by splitting, translating split components individually, and rejoining them.
  - This gives us an initial Hindi code version with basic translations.
- **Stage 2: GPT Enhancement**
  The second stage leverages GPT-4o Mini to refine and improve translations through example-based learning:
  - Using the Hindi code generated in Stage 1 output as reference examples for the model
  - Creating a customized prompt that shows the desired translation patterns to GPT model
  - We provide both the English code and the partially translated Hindi code from Stage 1
  - We use GPT to enhance the translation while preserving code functionality and structure
  - This generates a more natural and contexually appropriate Hindi version

1.4 Likely challenges and mitigations

There are several challenges in trying to translate Python code between English and Hindi. Our implementation tackles these through specific mitigation strategies:-

- **Compound Word Translation**: Code often contains compound words separated by underscores. The `translate_token` method in the `CodeTranslator` class handles this by splitting compound words, translating individual components, and rejoining them with underscores to maintain code readability.
- **Code Structure Preservation**: Maintaining code formatting and structure is crucial for readability and execution. The `translate_line` method preserves indentation and line structure by measuring leading whitespace before translation and restoring it afterward.
- **Translation Reliability**: To handle potential translation failures, we apply two important features in our code. The `safe_translate` method, which includes a retry mechanism for failed translations. Next is the `CheckpointManager` class which maintains translation progress. This allows longer translation tasks to be resumed if they are interrupted.
- **Keyword Translation**: Python keywords and built-in functions need consistent translation. The `KeywordManager` class maintains a dictionary mapping between English and Hindi keywords, which use Joshua Otten's curated dataset (Otten et al., 2021).
- **Comments and String Handling**: Code comments need a different handling than executable code. The `translate_line` method identifies comments by looking at the `'#'` character and applying separate translation rules for code and comment sections. This preserves the comments.

These challenges are addressed through specific implementation, though future work can definitely improve these solutions.

2 Related Work

3 Experiments

3.1 Datasets

Please list which datasets you used, whether or not you have access them, and whether or not they are publicly available with the same preprocessing and train / dev / tests as the previous work you will be comparing to (if applicable). If you plan to collect your own dataset for evaluating robustness, please describe clearly the data plan (the data source, how you plan to collect it, how you would preprocess it for the task, etc.).

3.2 Implementation

Please provide a link to a repo of your reimplementation (if applicable) and appropriately cite any resources you have used.

4 Evaluation

4.1 Dataset and Test Configuration

- **Total Dataset**: 500k Python code samples from hugging face (jtatman, 2024)
- **Test Configuration**:
  - Translation set: 10 samples selected for translation
  - Example sets: Varying sizes (5,10,20,30) that are given to GPT model for reference
  - Human evaluation set: 20 samples evaluated by human evaluators
- **Keyword Dictionary**: 234 pre-mapped English-Hindi keyword pairs translated by Joshua Otten.

4.2 Human Evaluation Framework

Two bilingual evaluators (Hindi-English) with 4 years of experience in Python programming evaluated 20 code samples generated by our model. They followed the following rating scale-

Rating Scale(1-5):

- **1**: Unusable and incorrect translation
- **2**: Partially correct translation, major revisions needed
- **3**: Mostly correct translation, minor revisions needed
- **4**: Good translations with minimal revisions needed
- **5**: Perfect translation, no revisions needed

Evaluation Criteria:

- **Syntax Correctness (SC)**: Proper keyword translation, code structure preservation and indentation accuracy
- **Semantic Preservation (SP)**: Logic preservation, variable scope maintenance and function behavior consistency
- **Hindi Language Quality (HLQ)**: Evaluating natural hindi expressions, the technical term consistency, comment clarity and code readability

4.3 Technical Evaluation Framework

Syntax Validation

- **AST(Abstract Syntax Tree) Validation**: Tests if translated code produces valid Python AST and gives a binary outcome of Valid/Invalid
  - Result: ????
- **Token Structure Analysis**: Compares token types between original and translated code, uses Python's tokenize module
  Similarity score: 0.6250 ????

Semantic Testing

We use the `SemanticTester` class to text the following:

- **Execution Equivalence**:Runs both original and translated code, compares outputs for identical inputs
  - Success rate: 0.5999 across test cases ????
- **Runtime Behavior**: Tests error handling, verifies output types and memory usage patterns

Translation Quality Metrics

- **BLEU Score Evaluation**:INSERT GRAPH HERE and Explain
- **Back Translation Validation**:INSERT GRAPH HERE and Explain

5 Results

Provide a table comparing your results to the published results.

5.1 Discussion

Discuss any issues you faced. Do your results differ from the published ones? If yes, why do you think that is? Did you do a sensitivity analysis (e.g. multiple runs with different random seeds)?

5.2 Resources

Discuss the cost of your reproduction in terms of resources: computation, time, people, development effort, communication with the authors (if applicable).

5.3 Error Analysis

Perform an error analysis on the model. Include at least 2-3 instances where the model fails. Discuss the error analysis in the paper – what other analyses could the authors have ran? If you were able to perform additional error analyses, report it here.

6 Robustness Study

Explain your approach for Evaluating the Model Robustness. Describe what robustness analysis you have performed. Provide sufficient details about your perturbation data, how you created it, how you used it as a robustness benchmark to evaluate the model, in what metrics, etc.

6.1 Results of Robustness Evaluation

Describe the evaluation results of your reproduced model on the robustness benchmark that you created. Include at least 2 examples where the model performs well and 2 examples where it fails (i.e., being not robust). Provide sufficient analysis and your thoughts on the observations.

6.2 Discussion

Provide any further discussion here, e.g., what challenges did you face when performing the analysis, and what could have been done if you will have more time on this project? Imagine you are writing this report to future researchers; be sure to include ”generalizable insights” (e.g., broadly speaking, any tips or advice you’d like to share for researchers trying to analyze the robustness of an NLP model).

7 Workload Clarification

Describe how the team divides the workload in this checkpoint. Note that each team member should contribute roughly the same amount of work to this assignment.

8 Conclusion

Is the paper reproducible?

References

jtatman. 2024. Python code dataset 500k. https://github.com/jtatman/python-code-dataset-500k. Accessed: 05-12-2024.

Joshua Otten, Antonios Anastasopoulos, and Kevin Moran. 2021. Unipy web tool. https://universal-pl.github.io/UniPy/. [Online; accessed 10-October-2023].