Python to Rust LLM translation

Guoyao Hao

Goal

- Using LLM, translate a multi-file Python project into a Rust project
- Keep Rust project a similar structure to the original project
- Evaluate generated rust code using AST structure similarity

The target python repo

- https://github.com/uiri/toml
- It’s written in pure Python.
- It has multiple Python files, and cross-file dependencies.
- Doesn’t rely on complex Python libraries.
- Medium size project.

Difficulty1

- Python is a dynamically-typed language. Rust, by contrast, is statically-typed.
- It could cause problems like
- 1. Missing type information
- 2. Generics versus duck typing
- 3. Rust using Option<T> for “nullable” fields
- and more

Solution: Infer variable types in Python

- Static Analysis tools
  Rely on the PEP 484 Type Hints
  Dynamic features leading to “Any” type

- Dynamic Analysis
  Work on all python code.
  Able to handle dynamic types
  Coverage problem.

- Tools I used
  Monkeytype
  sys.settrace (deprecated)

.pyi file with types

Use MonkeyType, dynamically run the code and get stub with types.

```python
def _detect_pathlib_path(p: TextIOWrapper): ...

def _ispath(p: TextIOWrapper) -> bool: ...

def _load_date(val: str) -> Union[date, datetime]: ...

def _load_unicode_escapes(v: str, hexbytes: List[str], prefix: str) -> str: ...

def _strictly_valid_num(n: str): ...

def _unescape(v: str) -> str: ...

class TomlDecoder:
    def __init__(self, _dict: Type[dict] = ...): ...
    def _get_split_on_quotes(self, line: str) -> List[str]: ...
    def _load_array_isstrarray(self, a: str): ...
    def _load_line_multiline_str(self, p: str) -> Tuple[int, int]: ...
    def bounded_string(self, s: str) -> bool: ...
    def embed_comments(self, idx: int, currentlevel: Dict[str, Any]): ...
    def get_empty_table(self) -> Dict[Any, Any]: ...
    def load_array(self, a: str) -> List[Union[int, str, List[int], List[Union[int, str]]]]: ...
    def load_line(
        self,
        line: str,
        currentlevel: Dict[str, Any],
        multikey: None,
        multibackslash: bool
    ) -> Tuple[str, str, bool]: ...
```

Dynamically infer variable types at runtime (Deprecated)

```
{
  "func": "TomlDecoder.load_line",
  "file_dir": "/Users/hao/Desktop/github/toml-python/toml/decoder.py",
  "lineno": 512,
  "args": {
    "self": "TomlDecoder",
    "line": "str",
    "currentlevel": "dict",
    "multikey": "NoneType",
    "multibackslash": "bool"
  },
  "return": "NoneType",
  "module": "toml",
  "file_name": "decoder"
},
```

```
{
  "func": "TomlDecoder.load_line",
  "file_dir": "/Users/hao/Desktop/github/toml-python/toml/decoder.py",
  "lineno": 680,
  "args": {
    "self": "TomlDecoder",
    "line": "str",
    "currentlevel": "DynamicInlineTableDict",
    "multikey": "bool",
    "multibackslash": "bool"
  },
  "return": "NoneType",
  "module": "toml",
  "file_name": "decoder"
},
```

```
"args": {
  "self": [
    "TomlDecoder"
  ],
  "line": [
    "str"
  ],
  "currentlevel": [
    "DynamicInlineTableDict",
    "dict"
  ],
  "multikey": [
    "bool",
    "NoneType"
  ],
  "multibackslash": [
    "bool"
  ]
},
"return": [
  "str",
  "bool",
  "NoneType"
]
},
```

- Union results from multiple function call

Decompose large task into smaller size

- Using Tree-sitter to parse the AST and build a dependency graph.
- In first iteration build CodeNode (could be class or function) as vertices for the Graph.
- In second iteration build edges based on dependency
  Caller -> Callee
  Child class -> Father class
  Method -> Class (action relies on the entity)

Difficulty2

- Python is dynamic attribute binding language. Rust is not
- To translate a Python class, it requires to explicitly declare every instance variable as a field in a Rust struct.
- That means to translate to a Rust Struct, we need information from Python functions.
- Method -> Class, Class -> Method. Cycle dependency!

Solution: Build Intermediate Representation

- Collect all instance variables, remove cycle dependency

```
{
  "qname": "TomlTz",
  "kind": "class",
  "filename": "tz.py",
  "lineno": 4,
  "body_ir": [
    "class TomlTz(tzinfo):",
    "def __init__(self, toml_offset):  # ...",
    "def __getinitargs__(self):  # ...",
    "def __deepcopy__(self, memo):  # ...",
    "def tzname(self, dt):  # ...",
    "def utcoffset(self, dt):  # ...",
    "def dst(self, dt):  # ..."
  ],
  "num_lines": 7,
  "fields": [
    "_raw_offset",
    "_sign",
    "_hours",
    "_minutes"
  ],
  "base_class": [
    "tzinfo"
  ]
}
```

Call LLM API

- Use LangChain with OpenAI API
- Generate base code (use std…. Define enum type)
- Generate all Structs in dependency order
- Generate code for all functions in dependency order
- Generate cargo.toml and lib.rs (manually)
- Write the generated code back to the graph structure

Prompt Design

- In addition to the python code,
- Add stub
- Add dependency (predecessors/successors)
- Add guideline
- Add parameter type and return type

```
    template="""
You are a Rust expert. Your task is to convert a Python function to Rust functions.
Only implement the target function, do not implement other functions.

You have the following stub signatures:
```python
{stub}
```


You have already generated:
```rust
{rust_accumulated}
```


The following functions have already been implemented, DO NOT implement these again:
{implemented_functions}

Translate this Python function to Rust, considering its dependencies:
```json
{dependencies}
```


Function: {qname}
Enclosing class: {class_name}
Parameters: {params}
Return type: {return_type}
Body IR:
{body_ir}
```

Click to collapse the range.

Evaluation AST similarity

- Different programming languages have different AST node types.
- For example, in python it's called function_definition
  in rust it's called function_item
- Direct comparison will cause bias
- Solution: rust_to_common; python_to_common
- Cosine similarity of Each CodeNode

Evaluation result

Similarity for each node, 4o-mini