

Generative Type Inference for Python

Yun Peng[†], Chaozheng Wang[†], Wenxuan Wang[†], Cuiyun Gao[‡], Michael R. Lyu[†]

[†] Computer Science and Engineering Department, The Chinese University of Hong Kong, Hong Kong, China

[‡] School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China

{ypeng, wxwang, lyu}@cse.cuhk.edu.hk, adf11178@gmail.com, gaocuiyun@hit.edu.cn

Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE2023)

発表者：松尾 和輝

背景

- 動的型付け言語であるPython人気の高まり
 - 人工知能やデータサイエンスのブーム
 - 利便性の高さや使いやすさ
 - コードスニペットの再利用による, 開発効率の向上
- 動的型付け言語は型エラーを起こしやすい
 - GH, SOに寄せられた質問の30%は型エラー関連^[30]
- 型エラーを減らすために型アノテーションを導入
 - 開発者らの負担が大きい

型アノテーションを自動で行う研究の増加

[30] Wonseok Oh and Hakjoo Oh. Pyter: effective program repair for python type errors. In Abhik Roychoudhury, Cristian Cadar, and Miryung Kim, editors, Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, Singapore, November 14-18, 2022, pages 922–934. ACM, 2022.

既存手法 (1/2)

- ルールベース型推論
 - ターゲットとなるコードを入力, ルールに基づき型を推論
 - 課題 | カバレッジの低さ
- 教師あり学習型推論
 - ターゲットとなるコードと周辺のコードを入力, ベクトル化し型を分類するモデルを学習
 - 課題 | 膨大な学習データが必要

既存手法 (2/2)

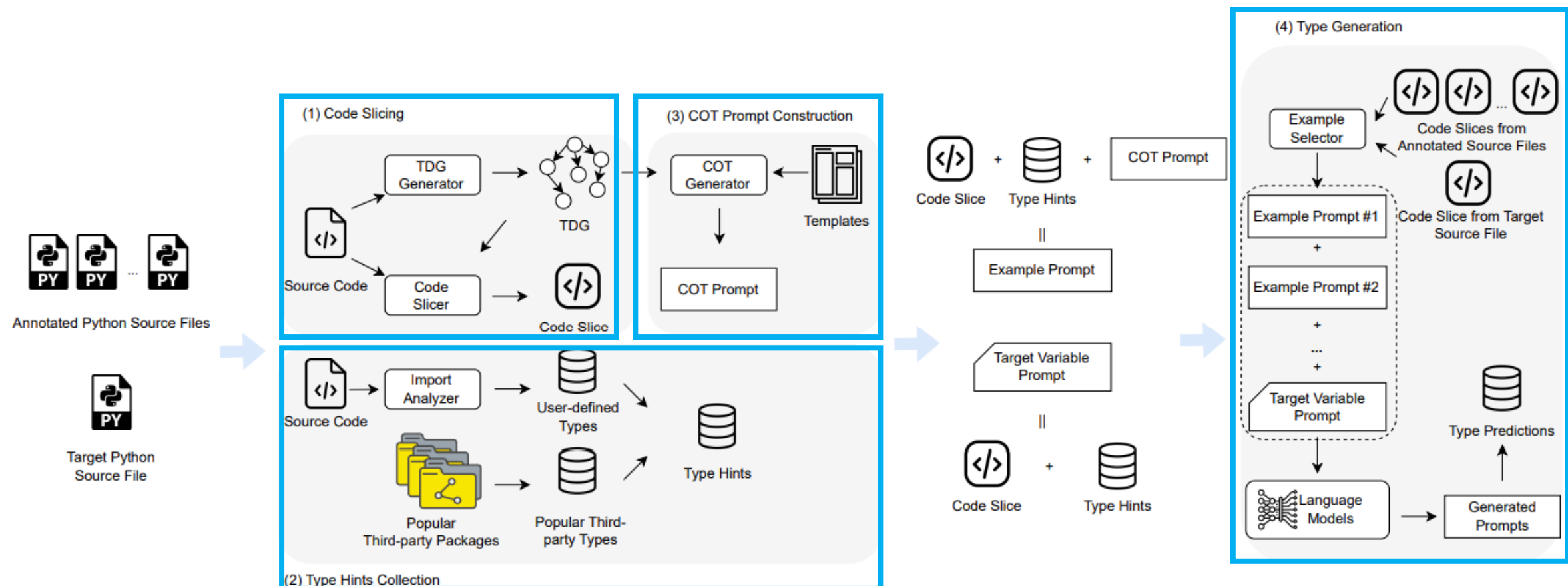
- Clozeスタイル型推論
 - 型の部分をマスクしたコードを入力, マスクされた部分をPTMで予測
 - 課題1 | 静的なドメイン知識の欠如
 - 課題2 | モデルの出力した予測が解釈できない

本研究の目的

Few-Shot生成型推論アプローチである**TYPEGEN**の提案

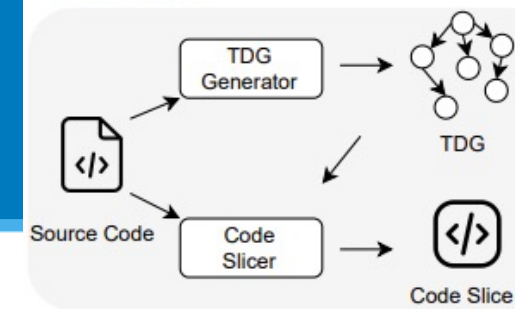
TYPEGEN | 概要図

- 以下の4STEPで構成
 - ① コードスライシング
 - ② 型ヒントの収集
 - ③ 思考連鎖型 (CoT) プロンプトの構築
 - ④ 型生成



①コードスライシング

(1) Code Slicing

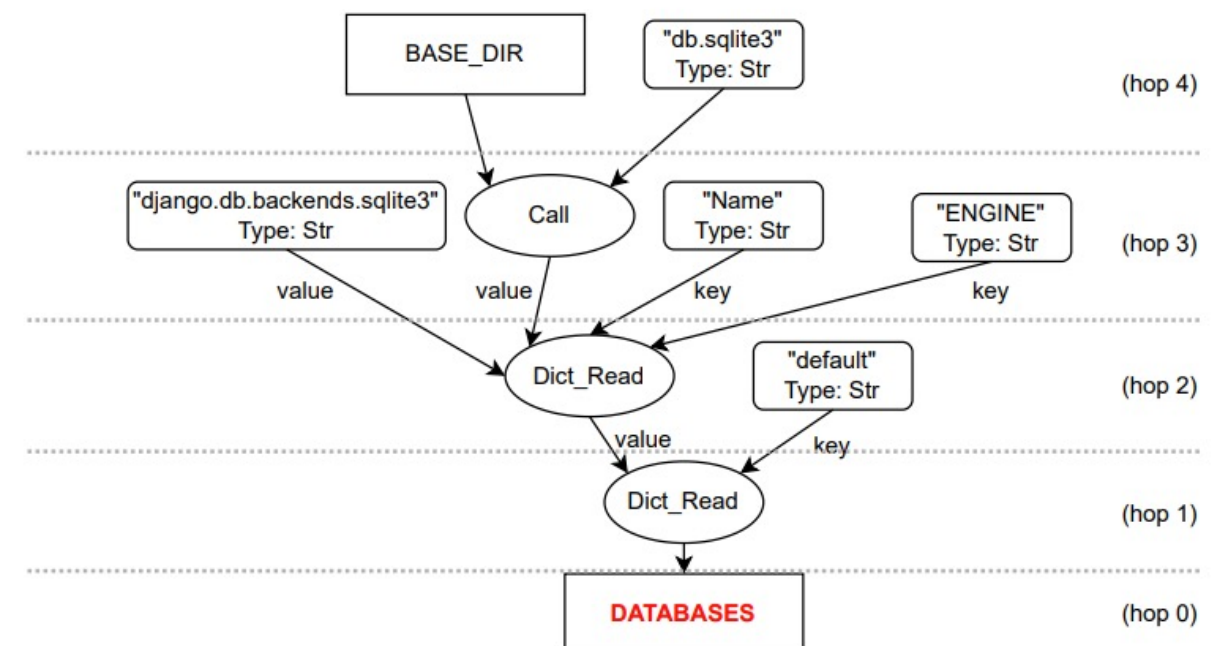
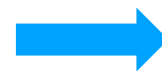


目的 | 変数間の型依存関係を示す型依存グラフ (TDG) [34]に基づき, ターゲット変数に関するコードを特定

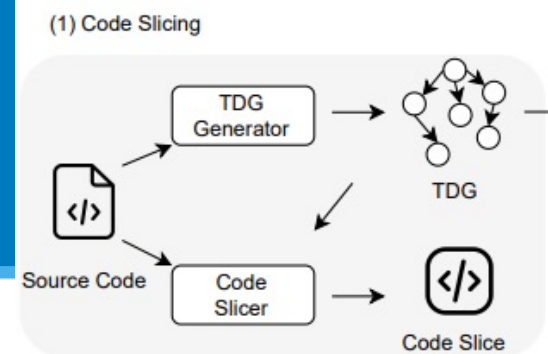
- TDG

- 変数 (Symbol), 操作 (Operation), 型 (Type) のノードを持つ有向グラフ
- HiTyper^[34]を用いて生成
 - コードをASTに変換後, どのノードに該当するか特定し変換?

```
...
12 import os
...
25 DEBUG = bool( os.environ.get('DJANGO_DEBUG', True) )
27 ALLOWED_HOSTS = ['stepper-v2.herokuapp.com', '127.0.0.1']
...
71 DATABASES = {
72     'default': {
73         'ENGINE': 'django.db.backends.sqlite3',
74         'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
75     }
76 }
...
129 db_from_env = dj_database_url.config(conn_max_age=500)
130 DATABASES['default'].update(db_from_env)
...
```



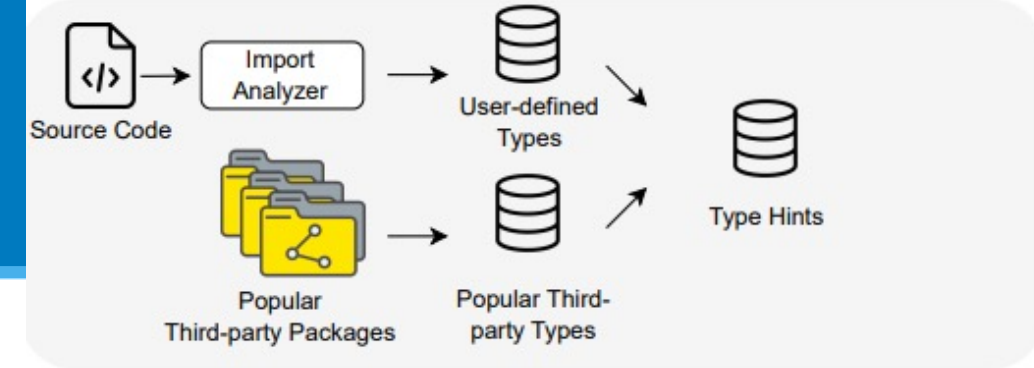
①コードスライシング



- ターゲット変数が関数の引数, 戻り値, ローカル変数
 - 特定の関数のTDGを生成
- ターゲット変数がグローバル変数
 - クラスと関数以外の全てのステートメントに対してTDGを生成
- TDG生成後は, ターゲット変数との型依存関係を持たないノードを刈り取る
- 刈り取り後のTDGから, ノードと対応するステートメントを結合しコードスライスを生成

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.sqlite3',  
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),  
    }  
}  
DATABASES['default'].update(db_from_env)
```

②型ヒントの収集



目的 | ユーザ定義型, サードパーティ型のへの対応

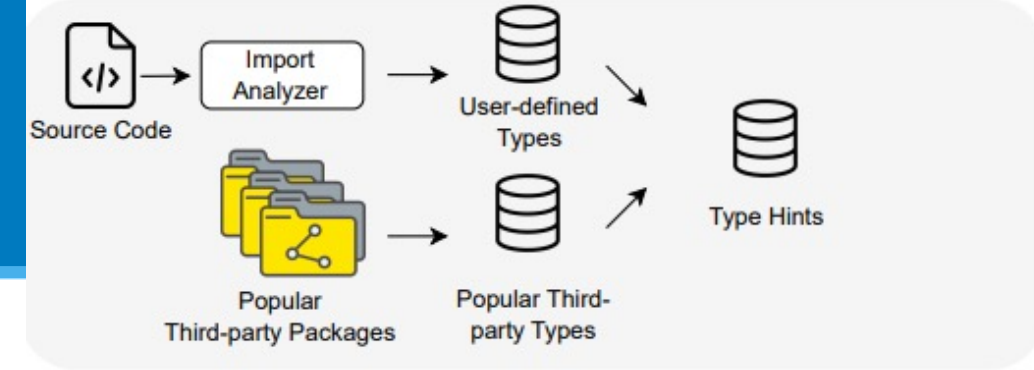
– ユーザ定義型の収集

- 現在のソースファイルからクラス定義をユーザ定義として収集
- Import文を調べどのソースファイルがインポートされているかを判断
- それらのクラス定義をユーザ定義として収集

– サードパーティ型の収集

- Libraries.io^[22]によりランク付けされた上位10,000の人気のPythonパッケージをダウンロード
- ユーザ定義型の収集と同様の処理を行い, サードパーティ型を収集
- 収集したサードパーティ型はデータベースに保存

②型ヒントの収集



- 型ヒントの生成
 - ユーザ定義型
 - 先述の通り
 - サードパーティ型
 - Import文からパッケージを特定
 - データベースに問い合わせ利用可能な全てのサードパーティ型を取得
 - 生成する型ヒントの数は50を上限とし，優先順位をつける

現在のソースファイルの
ユーザ定義型

>

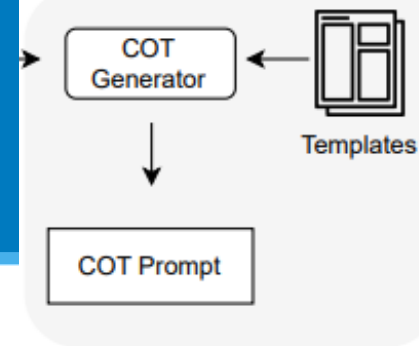
他のソースファイルの
ユーザ定義型

>

サードパーティ型

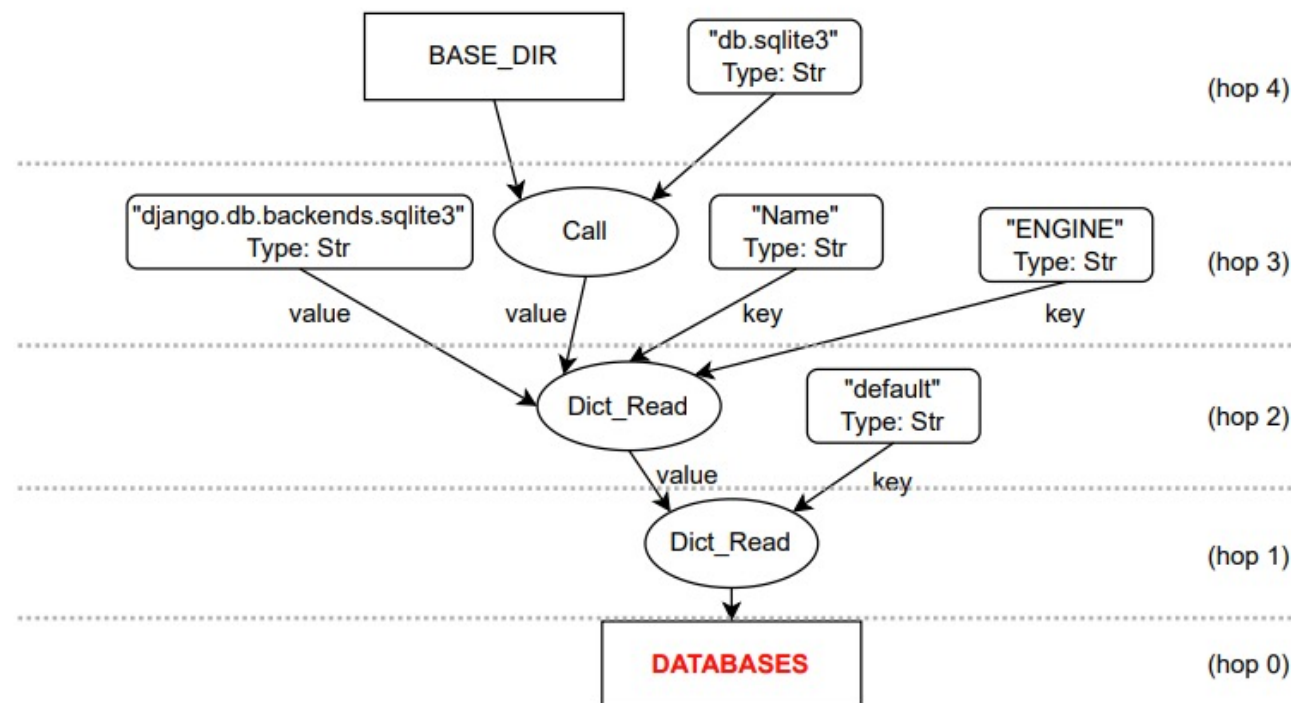
`os.Mapping, os.MutableMapping, os.PathLike, os._AddedDllDirectory, os._Environ, os._wrap_close`

③CoTプロンプトの構築

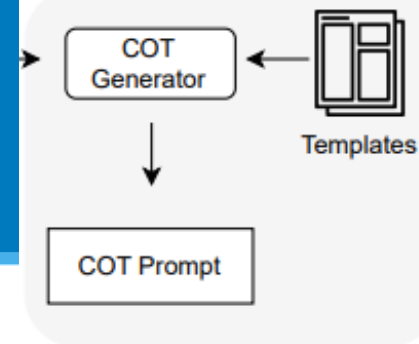


目的 | 型がどのように構築されるかの静的ドメイン知識に関与させる

- ①コードスライシングで得たTDGを入力とする
- ターゲット変数ノードからの距離に従って, ノードを異なるホップに整理



③CoTプロンプトの構築



- $\text{hop } i+1 \rightarrow \text{hop } i \quad (i \in \mathbb{N}^0)$ の関係を利用し，テンプレートによりプロンプトを自動生成

ローカル変数と
戻り値

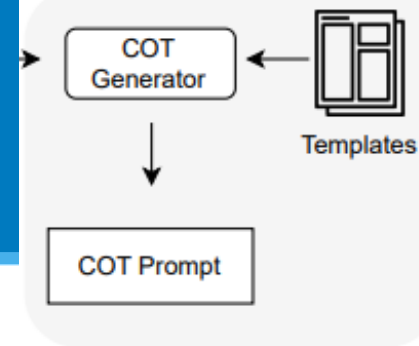
Part	Type	Template
DD-RV	Operation→Symbol	The variable/return value of [NAME] is assigned from [OP] operation.
	Symbol→Symbol	The variable/return value of [NAME] is assigned from variable [NAME].
	Type→Symbol	The variable/return value of [NAME] is assigned from [TYPE].
	Operation→Operation	The operand(s)/target(s)/key(s)/value(s) of [OP] is/are [OP] operation.
	Symbol→Operation	The operand(s)/target(s)/key(s)/value(s) of [OP] is/are variable [NAME].
	Type→Operation	The operand(s)/target(s)/key(s)/value(s) of [OP] is/are [TYPE].
DD-A	Usage	The argument [NAME] is used in [OP]/[NAME].
	Naming	Based on the naming convention, it is reasonable to assume that the type of the argument [NAME] is [GTTYPE].
Con	Conclusion	Therefore, the type of the variable/return value of/argument [NAME] is [GTTYPE].

引数

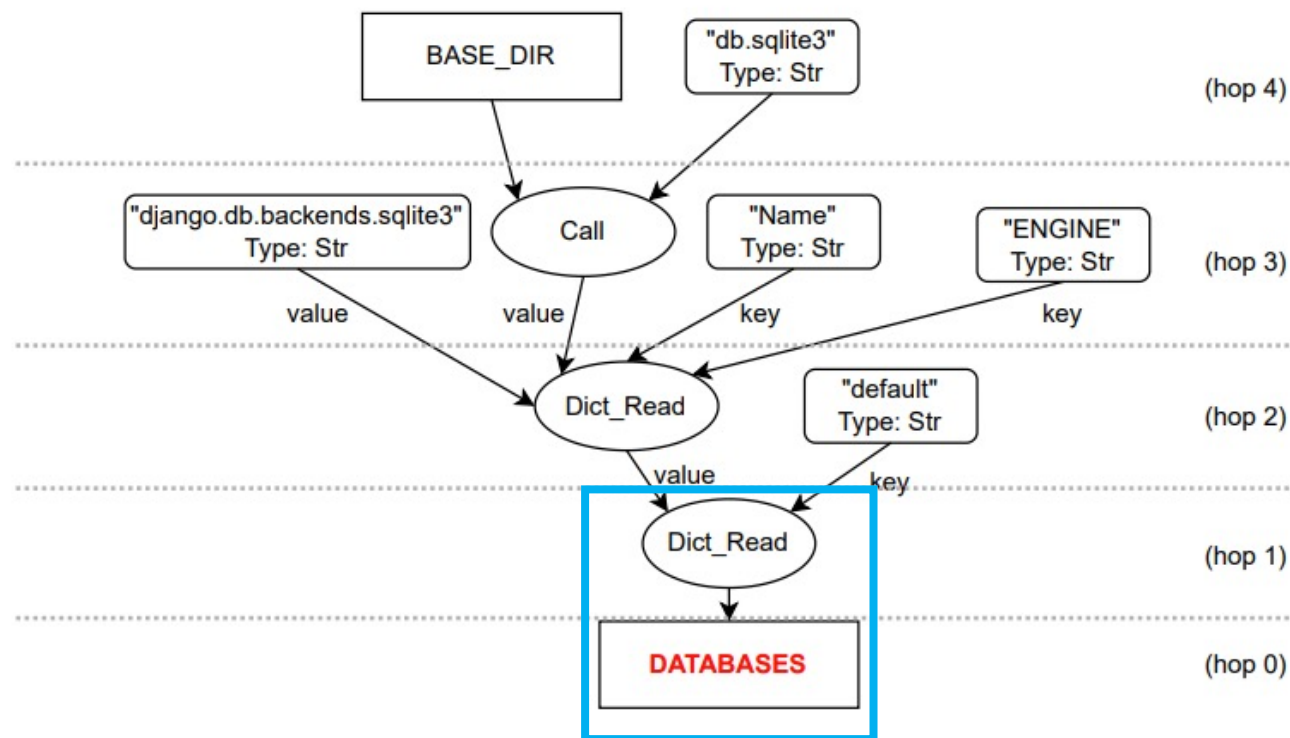
結論

[NAME] : Symbolノード名
 [OP] : Operationノード名
 [TYPE] : Typeノード名
 [GTTYPE] : ターゲット変数の型

③CoTプロンプトの構築



• 例



Part	Type	Template
DD-RV	Operation→Symbol	The variable/return value of [NAME] is assigned from [OP] operation.
	Symbol→Symbol	The variable/return value of [NAME] is assigned from variable [NAME].
	Type→Symbol	The variable/return value of [NAME] is assigned from [TYPE].
	Operation→Operation	The operand(s)/target(s)/key(s)/value(s) of [OP] is/are [OP] operation.
	Symbol→Operation	The operand(s)/target(s)/key(s)/value(s) of [OP] is/are variable [NAME].
	Type→Operation	The operand(s)/target(s)/key(s)/value(s) of [OP] is/are [TYPE].
DD-A	Usage	The argument [NAME] is used in [OP]/[NAME].
	Naming	Based on the naming convention, it is reasonable to assume that the type of the argument [NAME] is [GTTYPE].
Con	Conclusion	Therefore, the type of the variable/return value of/argument [NAME] is [GTTYPE].

生成文：The variable DATABASES is assigned from a dict.

A: First, the variable DATABASES is assigned from a dict. **Second**, the key of the dict is a str. The value of the dict is a dict. **Third**, the keys of the dict are a str and a str. The values of the dict are a str and a function call os.path.join. **Therefore**, the type of the variable DATABASES is `'dict[str, dict[str, str]]'`.

生成されるプロンプト

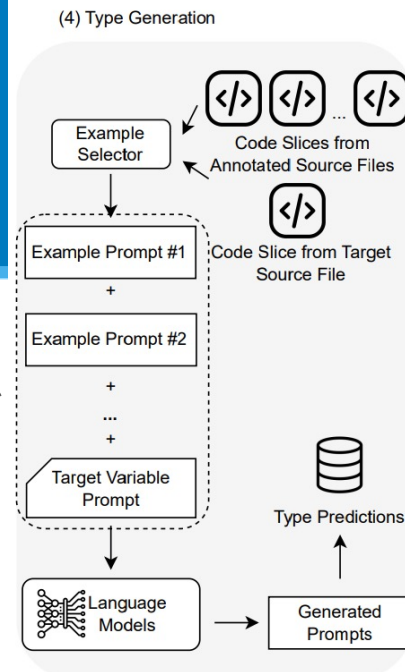
- ①②③で生成されたプロンプトを結合し，例となるプロンプトを作成
- 複数のプロンプトを作成し，学習セットとする

Example Prompt:

①. Code Slice	Python Code: <pre>DATABASES = { 'default': { 'ENGINE': 'django.db.backends.sqlite3', 'NAME': os.path.join(BASE_DIR, 'db.sqlite3'), } } DATABASES['default'].update(db_from_env)</pre>
②. Type Hint	Available User-defined Types: os.Mapping, os.MutableMapping, os.PathLike, os._AddedDllDirectory, os._Environ, os._wrap_close Q: What's the type of the variable DATABASES?
③. COT Prompt	A: First , the variable DATABASES is assigned from a dict. Second , the key of the dict is a str. The value of the dict is a dict. Third , the keys of the dict are a str and a str. The values of the dict are a str and a function call os.path.join. Therefore , the type of the variable DATABASES is <code>'dict[str, dict[str, str]]'</code> .

④型生成

- 入力ソースファイルとターゲット変数に対してコードスライスを生成
- BM25類似度^[40]に基づき，学習セットから類似するプロンプトを選択
- 複数の例からターゲット変数に対する同様のCoTプロンプトを生成するように学習
 - 型の部分を引用符で囲むようにしている
- 使用モデルは
 - GPT-Neo^[3]， GPT-J^[43]， CodeGen^[29]， GPT3.5^[4]， ChatGPT^[31]



Target Variable Prompt:	
①. Code Slice	Python Code: [Code]
②. Type Hint	Available User-defined Types: [User-defined types from static analysis]
	Q: What's the type of the variable [name]?
	A: [To be generated]

RQと結果

- RQ1：TYPEGENの有効性
 - SoTAよりも8.6%精度が向上
- RQ2：異なる言語モデルにおけるTYPEGENの能力
 - 一貫して改善可能
- RQ3：アブレーションスタディ
 - コードスライシングの影響が大きい
- RQ4：例の数の違いによる影響
 - 5つの時最高の性能を示し，それ以降は低下した（1%程）

まとめと所感

- まとめ

- Python型推論手法であるTYPEGENの提案
- 既存手法の課題を解決しつつ，精度向上

- 所感

- TDGへの変換方法とか，コードスライスへの変換方法とか，インポート解析とか細々とした部分がよくわからない
- CoTプロンプトを活用している手法を初めて見たかも
 - テンプレートの作成には手間がかかりそうやけどすごく有用そう
- 結局どういうプロンプトが最適なんでしょう？