Generative Type Inference for Python

Yun Peng, Chaozheng Wang,
Wenxuan Wang, Cuiyun Gao *, Michael R. Lyu

ASE'23

香港中文大學
The Chinese University of Hong Kong

哈爾濱工業大學(深圳)
HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

Type Inference

```
def add(num1, num2):
    a = num1 + num2
    b = 1 + 2
    return a + b
```

Parameters:

num1 : ?

num2 : ?

Local Variables:

a : ?

b : ?

Return Value:

add : ?

Static Type Inference

```
def add(num1, num2):
    a = num1 + num2
    b = 1 + 2
    return a + b
```

$$\frac{}{\pi \vdash 1 : \text{int}} \qquad \frac{}{\pi \vdash 2 : \text{int}} \qquad \text{(Constant)}$$

$$\frac{\pi \vdash 1 : \text{int} \qquad \pi \vdash 2 : \text{int}}{\pi \vdash 1 + 2 : \text{int}} \qquad \text{(Add)}$$

$$\frac{\pi \vdash 1 + 2 : \text{int}}{\pi \vdash d : \text{int}} \qquad \text{(Assign)}$$

$$\frac{\text{Premise1, …, PremiseN}}{\text{conclusion}}$$

Static Type Inference

```
def add(num1, num2):
        a = num1 + num2
        b = 1 + 2
        return a + b
```

Parameters:

num1 : ?

num2 : ?

Local Variables:

a : ?

b : int

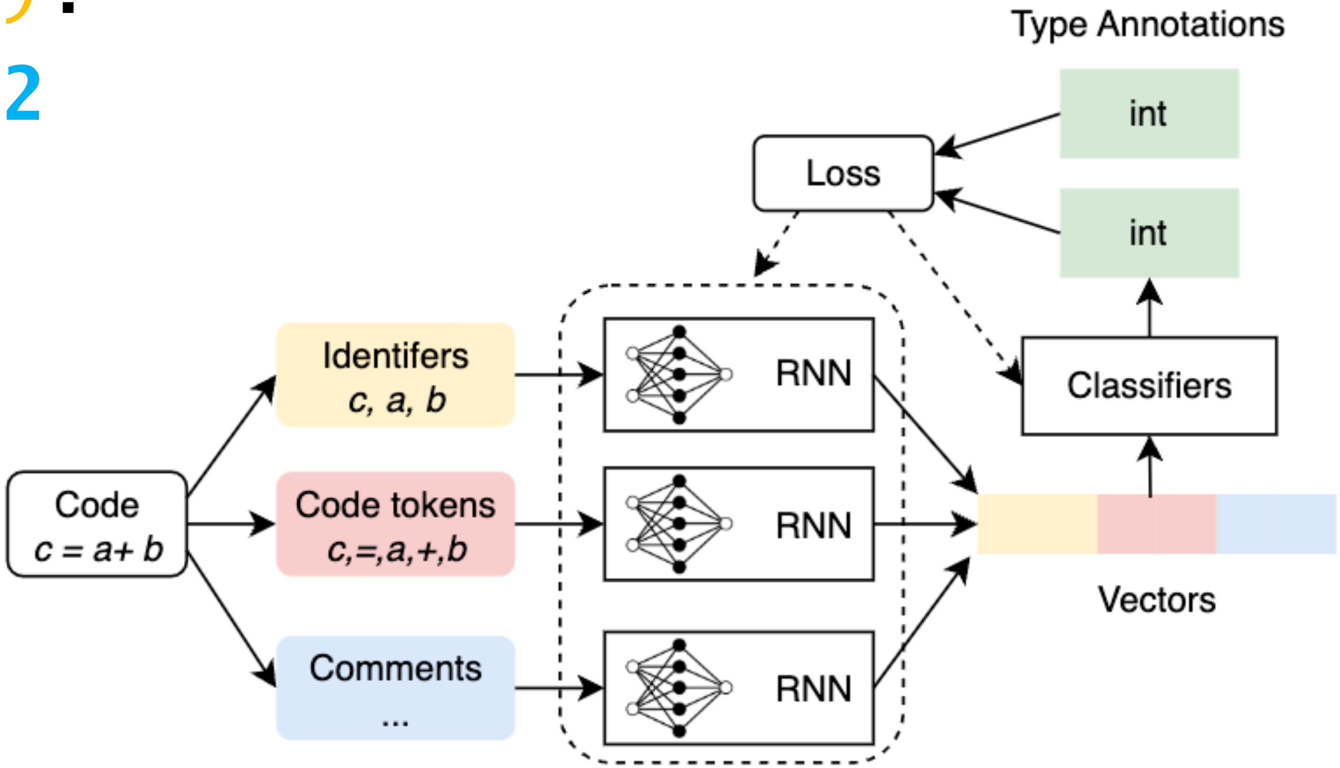
Return Value:

add : ?

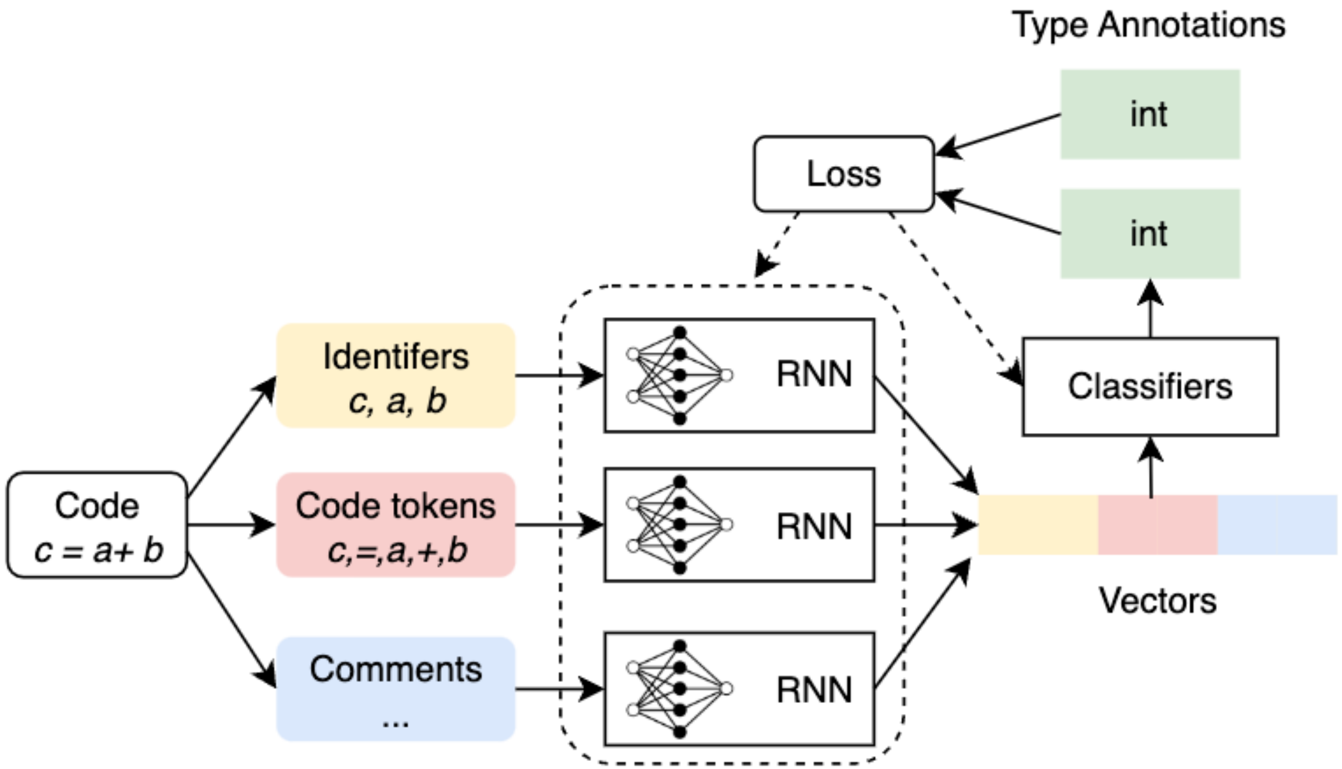
- Very accurate (sound)
- Suffer from the low coverage problem

Supervised Type Inference

```
def add(num1, num2):
    a = num1 + num2
    b = 1 + 2
    return a + b
```



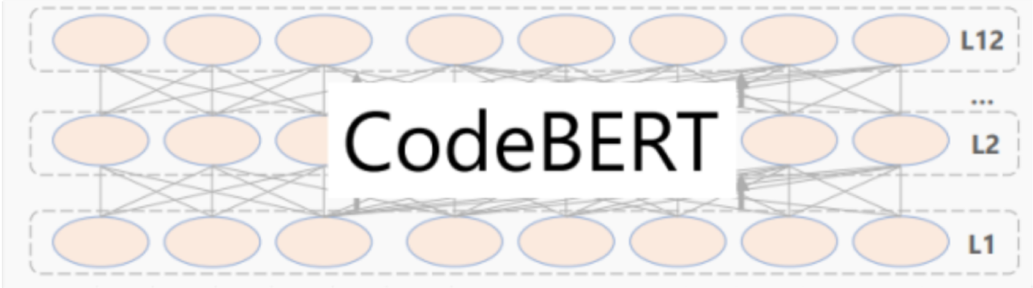
Supervised Type Inference



- Address the low coverage problem
- Require high-quality type annotations to train

Cloze-Style Type Inference

```
def add(num1:<mask0>, num2:<mask1>) -> <mask2>:
    c:<mask3> = num1 + num2
    d:<mask4> = 1 + 2
    return c + d
```



Parameters:

<mask0> (num1) : int

<mask1> (num2) : int

Local Variables:

<mask3> (c) : int

<mask4> (d) : int

Return Value:

<mask2> (add) : int

Cloze-Style Type Inference

```
def add(num1:<mask0>, num2:<mask1>) -> <mask2>:
    c:<mask3> = num1 + num2
    d:<mask4> = 1 + 2
    return c + d
```

- Do not require a high quality training set
- Lack of static domain knowledge:
  With knowledge only in the pre-trained code models
- Lack of interpretability:
  No idea how the model reaches the prediction

TypeGen: Generative Type Inference

Input prompt with static domain knowledge

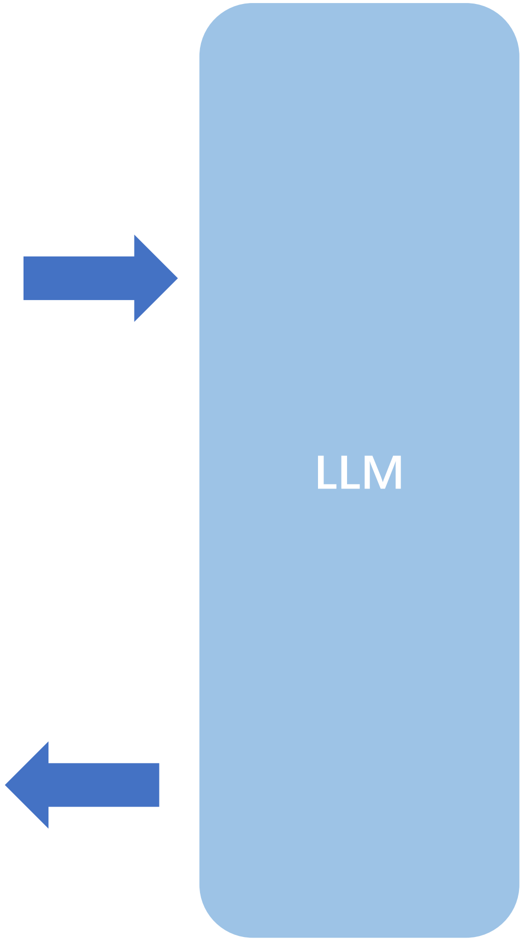
Python Code:

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

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?



Output chain-of-thought prompt making predictions

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]]`.

Let LLMs act like a static type inference tool!

See what static inference sees, think how static inference thinks.

TypeGen: Generative Type Inference

Challenge 1: Lack of static domain knowledge

What knowledge should a model have to infer a type for a variable? (See what static inference sees)

Knowledge 1: The context of the target variable

Parameters, return value, and local variables are defined based on functions. Therefore, the entire function can be the context.

Intuitive!

The Locality of Type Inference

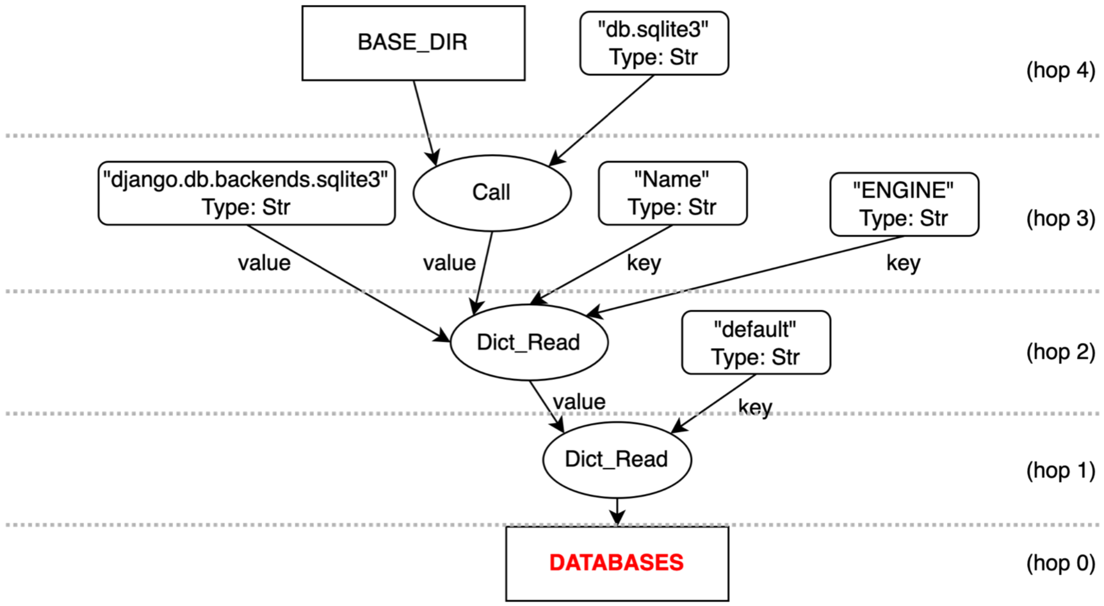
```
...
import os
...

...
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
...
db_from_env = dj_database_url.config(conn_max_age=500)
DATABASES['default'].update(db_from_env)
...
```

However, not all statements in the function are related to the target variable.

Step 1: Code Slicing

```
...
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)
...
```

Source Code



Type Dependency Graph

Step 1: Code Slicing

- Remove all statements without data dependencies with the target variable.
- Remove statements with very far data dependencies with the target variable.

```
...
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)
...
```

Original Code

```
...
71 DATABASES = {
72    'default': {
73        'ENGINE': 'django.db.backends.sqlite3',
74        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
75    }
76 }

130 DATABASES['default'].update(db_from_env)
...
```

Sliced Code

TypeGen: Generative Type Inference

Challenge 1: Lack of static domain knowledge

What knowledge should a model have to infer a type for a variable?
(See what static inference sees)

Knowledge 1: The context of the variable

Knowledge 2: The valid type set of the variable

Valid type set = built-in types + **imported types?**

Step 2: Type Hints Collection

Imported types = third-party types + user-defined types

User-defined types:

- Collect all classes in the current source file.

Third-party types:

- Download top 10,000 popular Python packages in Libraries.io.
- Collect all classes and their paths as a third-party type database.
- Query the database based on the import statements in current source file.

TypeGen: Generative Type Inference

Challenge 1: Lack of static domain knowledge

What knowledge should a model have to infer a type for a variable?
(See what static inference sees)

Knowledge 1: The context of the variable
Knowledge 2: The valid type set of the variable

Python Code:

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

Available User-defined Types:

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

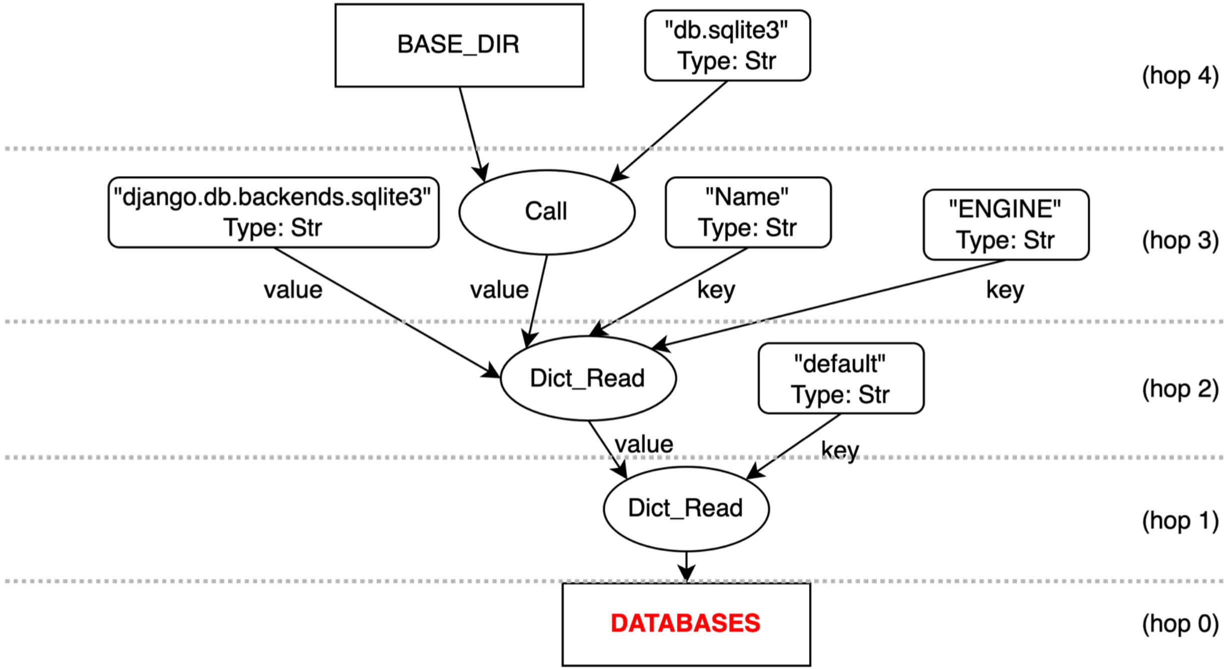
TypeGen: Generative Type Inference

Challenge 2: Lack of Interpretability

How to know/guide the model to reach a type prediction like static inference?
(Think how static inference thinks)

Simulate the inference steps of static inference!

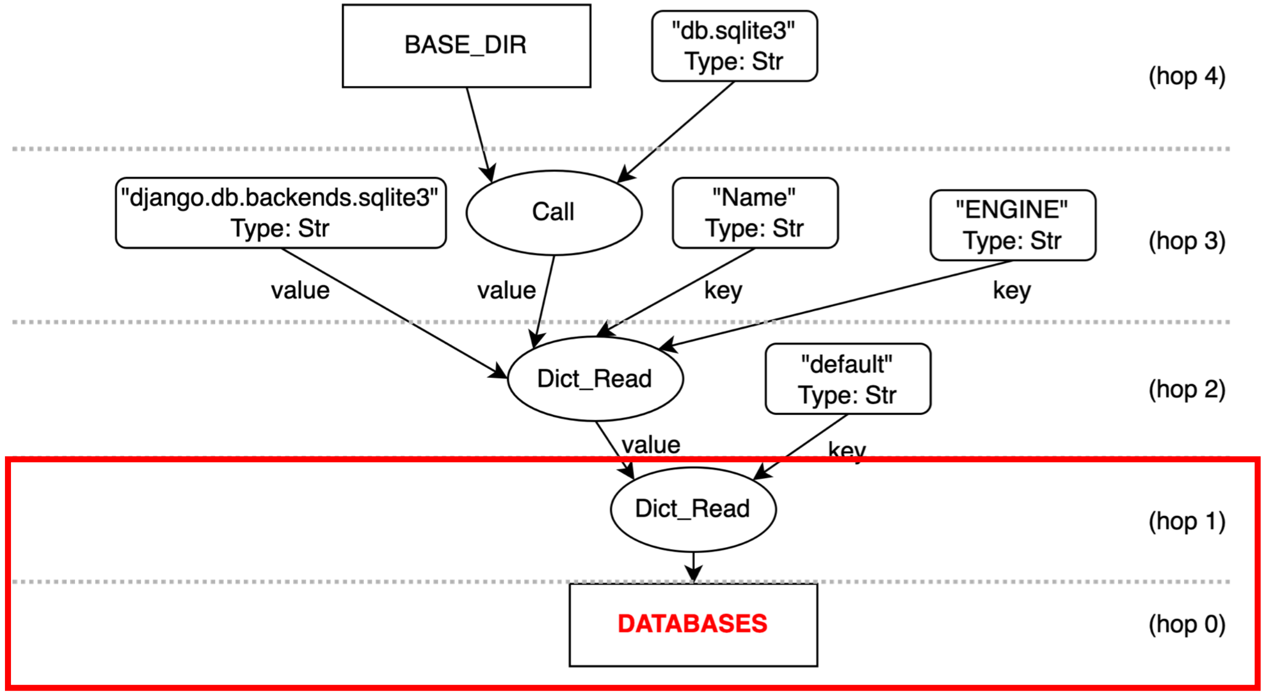
Step 3: Chain-of-Thought Prompt Generation



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].

Translate the Type Dependency Graph into a Chain-of-Thought prompt.

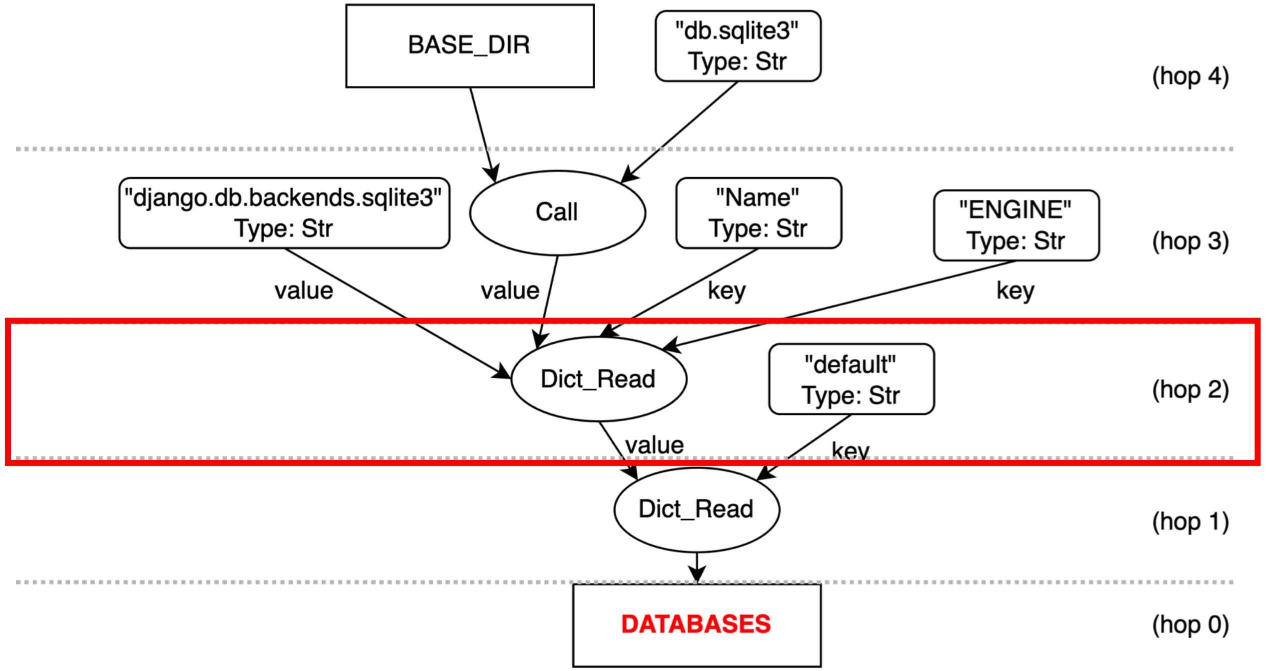
Step 3: Chain-of-Thought Prompt Generation



First, the variable DATABASES is assigned from a dict.

Translate the Type Dependency Graph into a Chain-of-Thought prompt.

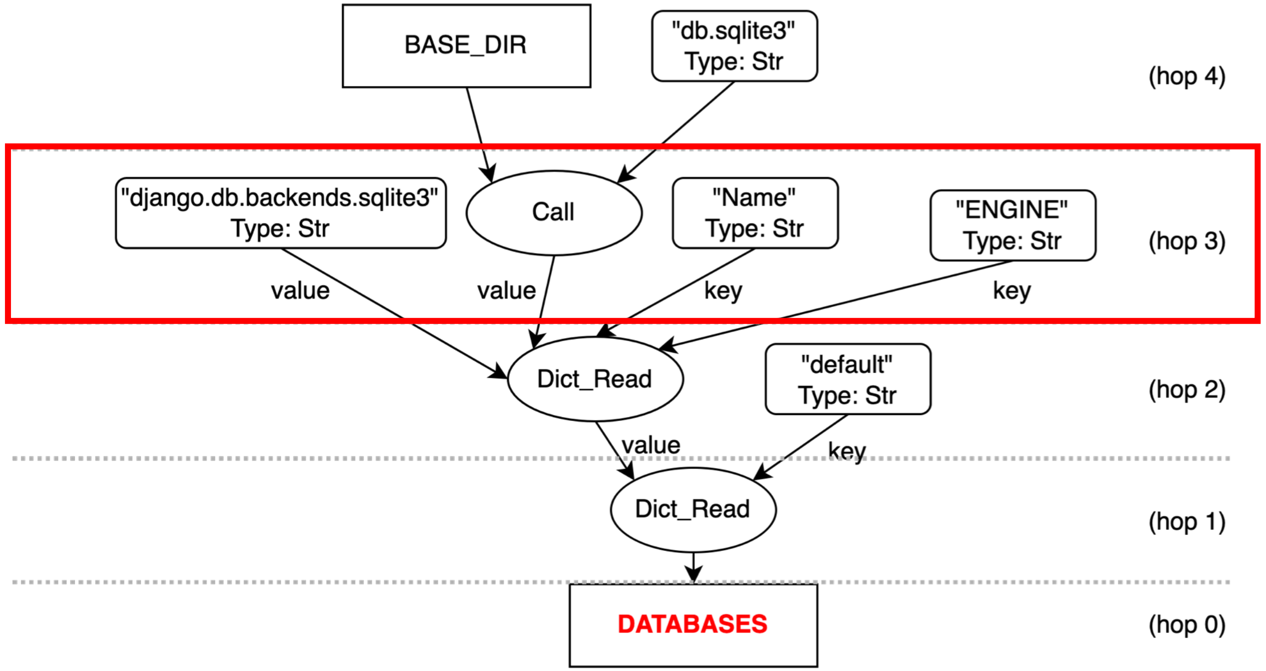
Step 3: Chain-of-Thought Prompt Generation



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.

Translate the Type Dependency Graph into a Chain-of-Thought prompt.

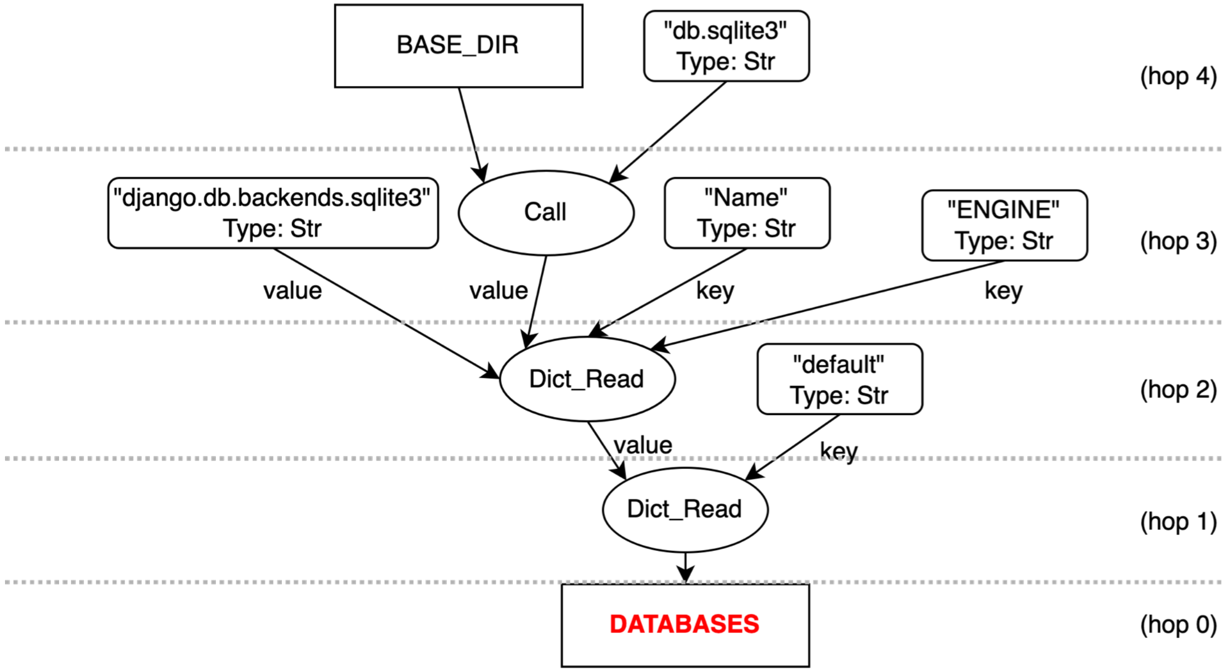
Step 3: Chain-of-Thought Prompt Generation



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.

Translate the Type Dependency Graph into a Chain-of-Thought prompt.

Step 3: Chain-of-Thought Prompt Generation



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]]`.

Translate the Type Dependency Graph into a Chain-of-Thought prompt.

Put Them All Together…

①. Code Slice

Python Code:

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

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 `dict[str, dict[str, str]]`.

In-Context Learning

Example Prompt:

Static Analysis Generated

①. Code Slice

Python Code:

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

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 `dict[str, dict[str, str]]`.

Target Variable Prompt: +

LLM Predicted

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]**

Performance of TypeGen

Metric	Category	Approach	Top-1				Top-3				Top-5			
			Arg	Ret	Var	All	Arg	Ret	Var	All	Arg	Ret	Var	All
Exact Match (%)	Supervised	TypeBERT	28.0	38.5	51.1	45.4	34.8	52.6	55.8	51.4	36.5	57.1	58.6	54.1
		TypeWriter	53.3	52.8	-	-	61.1	60.7	-	-	65.8	65.3	-	-
		Type4Py	66.5	56.1	82.0	76.6	72.0	59.2	83.8	79.3	73.8	60.7	84.3	80.1
	Cloze Style	InCoder-1.3B	20.9	20.5	15.1	16.7	21.3	20.8	15.5	17.1	21.3	21.0	15.6	17.2
		InCoder-6.7B	24.1	42.0	18.7	21.9	24.6	42.7	19.1	22.3	24.7	43.1	19.2	22.4
		UniXcoder	55.0	49.2	35.9	40.9	66.9	64.6	42.1	49.0	70.6	69.8	45.2	52.4
		CodeT5-base	51.1	57.6	21.7	30.7	59.3	64.4	28.0	37.4	62.0	66.9	30.7	40.1
		CodeT5-large	56.2	60.2	44.7	48.4	61.6	64.5	50.4	53.9	63.9	66.3	53.4	56.6
	Generative	TypeGen	73.1	68.7	82.2	79.2	81.0	77.1	87.9	85.6	82.7	79.1	89.1	87.0
Match to Parametric (%)	Supervised	TypeBERT	29.8	41.4	54.0	48.1	36.0	55.9	58.0	53.5	37.7	60.8	61.2	56.5
		TypeWriter	54.4	54.1	-	-	63.4	63.5	-	-	68.8	69.3	-	-
		Type4Py	68.0	59.0	86.2	80.2	74.1	64.1	88.3	83.3	75.9	66.3	88.8	84.3
	Cloze Style	InCoder-1.3B	22.9	22.8	18.7	19.9	23.3	23.1	19.1	20.3	23.4	23.3	19.2	20.4
		InCoder-6.7B	28.8	51.6	25.0	28.1	29.3	52.1	25.3	28.5	29.4	52.5	25.3	28.6
		UniXcoder	61.9	61.8	44.3	49.3	72.3	76.0	51.2	57.6	75.0	80.1	53.8	60.4
		CodeT5-base	54.8	66.7	27.7	36.6	62.9	74.2	34.4	43.6	65.6	76.4	37.1	46.3
		CodeT5-large	61.4	69.4	55.7	58.0	66.8	74.3	61.2	63.5	68.9	76.2	63.7	65.9
	Generative	TypeGen	78.7	75.6	91.2	87.3	84.9	83.0	93.7	91.0	86.1	84.5	94.1	91.7

Performance of TypeGen

Base Model	Approach	Top-1 (△)	Top-3 (△)	Top-5 (△)
GPT-Neo (1.3B)	Zero-Shot	31.5	40.6	42.8
	Standard ICL	44.0 (40%)	50.0 (23%)	50.8 (19%)
	TypeGen	57.0 (81%)	61.5 (51%)	62.8 (47%)
GPT-Neo (2.7B)	Zero-Shot	43.2	50.0	51.9
	Standard ICL	46.6 (8%)	52.3 (5%)	52.8 (2%)
	TypeGen	55.5 (28%)	61.9 (24%)	63.0 (21%)
GPT-J (6.7B)	Zero-Shot	42.4	43.7	43.9
	Standard ICL	50.8 (20%)	54.9 (26%)	55.3 (26%)
	TypeGen	62.7 (48%)	67.3 (54%)	68.4 (56%)
CodeGen (6B)	Zero-Shot	34.7	44.0	45.5
	Standard ICL	54.1 (56%)	60.5 (38%)	61.9 (36%)
	TypeGen	63.7 (84%)	69.1 (57%)	70.8 (56%)
GPT-3.5 (175B)	Zero-Shot	62.0	65.4	66.3
	Standard ICL	69.7 (12%)	74.2 (13%)	75.8 (14%)
	TypeGen	78.9 (27%)	85.0 (30%)	86.2 (30%)
ChatGPT (175B)	Zero-Shot	61.3	66.1	67.5
	Standard ICL	68.0 (11%)	71.8 (9%)	73.1 (8%)
	TypeGen	78.8 (29%)	85.3 (29%)	86.7 (28%)

Ablation	Arg	Ret	Var	Ele	Gen	Usr	All
w/o Code Slice	74.8	77.0	68.8	75.1	75.5	73.9	70.8
w/o Type Hint	76.1	75.9	89.3	94.1	77.2	75.9	85.5
w/o COT Prompt	82.3	78.6	86.4	92.9	70.8	84.3	84.9
TypeGen	83.5	79.4	89.7	94.3	77.8	84.6	87.5

TypeGen is capable of **consistently improving** the zero-shot performance of type inference for language models **with different parameter sizes** and achieves **2x ~ 3x** of improvements made by the Standard ICL setting.

Conclusion

Input prompt with static domain knowledge

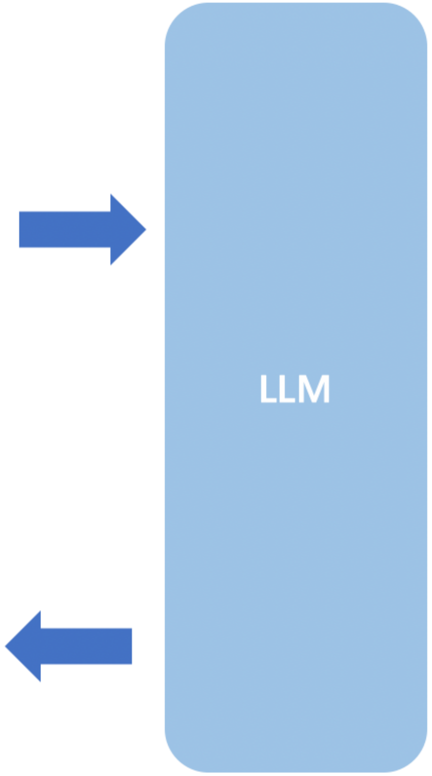
Python Code:

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

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?



Output chain-of-thought prompt making predictions

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]]`.

Let LLMs act like a static type inference tool!

See what static inference sees, think how static inference thinks.