# Database:

###### 数据：

1. self.library\_list(list[LIBRARY]): 保存本数据库涉及的所有LIBRARY。
2. self.threshold(float): 保存本数据库本次的阈值
3. self.mode(int): 模式，模式1为FUZZING，模式2为DIFFERENCE TESTING

--- 以上数据从配置文件中读取 ---

1. self.implicit\_layer\_info(dict[LIBRARY: dict[IMPLICIT\_LAYER\_NAME: dict(LAYER\_INFO)]])：一张表，维护self.library\_list中各LIBRARY下所有具体api的layer\_info。

该表会在初始化时读取，LIBRARY\_LIST更新时重新读取

1. self.implicit\_layer\_similarity(dict[LIBRARY: dict[IMPLICIT\_API\_NAME: dict[IAN: IAN]]])：一张表，维护self.library\_list中各LIBRARY下所有具体api的layer\_similarity。

该表会在初始化时读取，LIBRARY\_LIST更新时重新读取

1. self.implicit\_layer\_similarity\_valid(格式同上)：由上表基于THRESHOLD计算而得。在字典中，去掉未到阈值的项，去掉形状不合的项，去掉自身项，得到一个有效的变异表。

该表会在初始化时计算，并在THRESHOLD更新时重新计算。

1. self.api\_name\_map(dict[ABS\_API\_NAME: dict[LIBRARY: IMPLICIT\_API\_NAME]])：一个abstract\_api\_name到implicit\_api\_name的映射表，它从csv文件中直接读取，且只会读取与self.library\_list相关的api。

该表会在初始化时读取，LIBRARY\_LIST更新时重新读取

1. self.api\_name\_map\_valid(dict[ABS\_API\_NAME: dict[LIBRARY: IMPLICIT\_API\_NAME]])：一个有效的映射表，由上表计算得来。有效的定义为：所有在library\_list中的LIBRARY下均有映射，则该abstract\_api\_name就是有效的。

该表会在初始化时计算，LIBRARY\_LIST更新时重新计算

1. self.inverse\_api\_name\_map(dict[LIBRARY: dict[IMPLICIT\_API\_NAME: ABS\_API\_NAME]])：一个反映射表，由self.api\_name\_map\_valid计算得来。

该表会在初始化时计算，LIBRARY\_LIST更新时重新计算

1. self.inverse\_api\_name\_map\_valid(dict[LIBRARY: dict[IAN: ABS\_API\_NAME]])

一个反映射表，这次该表只会包含到有效的abstract\_api\_name

该表会在初始化时计算，LIBRARY\_LIST更新时重新计算

1. self.all\_api\_list(dict[LIB/ABS: list[ABS\_API\_NAME/IMPLICIT\_API\_NAME]])

保存所有涉及到的api\_name，主要用于合法性检查。

该表会在初始化时计算，LIBRARY\_LIST更新时重新计算

1. self.candidate\_map(dict[ABS\_API\_NAME: list[ABS\_API\_NAME]])：用于MODE2中的API\_MUTATE，基于THRESHOLD提供可变异的范围。

该表会在初始化时计算，并在THRESHOLD更新时重新计算。

1. self.api\_para\_map(dict[ABS\_API\_NAME: dict[ABS\_PARA\_NAME: dict[LIB: IPN]]])：一个(abstract\_api\_name, abstract\_para\_name, library) 到 implicit\_para\_name的映射表。

（暂时）在 MODE=1时候，会直接一对一完全映射

# TODO

1. self.abstract\_api\_layer\_info(dict[ABS\_API\_NAME: dict[LAYER\_INFO]])

（暂时）在MODE=1时候，会直接使用对应框架的LAYER\_INFO

# TODO

###### **方法**：

**有效性验证：**

is\_library\_valid(self, library: str) -> bool

验证一个library是否有效，即是否在本数据库的library\_list中

is\_abstract\_api\_name\_valid(self, abstract\_api\_name: str) -> bool

验证一个abstract\_api\_name是否有效。

is\_implicit\_api\_name\_valid(self, library: str, implicit\_api\_name: str) -> bool

验证在一个library下，implicit\_api\_name是否有效

**信息接口：**

get\_implicit\_api\_similarity\_valid(self, library: str, implicit\_api\_name: str) -> dict

获取对应LIBRARY对应IMPLICIT\_API\_NAME的相似度字典（有效的）

get\_implicit\_layer\_info(self, library: str, implicit\_api\_name: str) -> dict

获取对应library下对应的implicit\_api\_name的layer\_info字典

get\_implicit\_api\_name(self, library: str, abstract\_api\_name: str) -> str

获取对应abstract\_api\_name在library下映射到的implicit\_api\_name

get\_abstract\_api\_name(self, library: str, implicit\_api\_name: str) -> str

获取对应library下的implicit\_api\_name反映射到的abstract\_api\_name

get\_candidate\_mutate\_list(self, abstract\_api\_name: str) -> list[str]

获取对应abstract\_api\_name的可变异范围

generate\_candidate\_report(self, threshold: float = None) -> None

生成一个报告，查看在当前数据库条件下各个abstract\_api\_name的可变异范围。报告会放入report文件夹中。如果不提供threshold，会生成0.9~0.1的报告。

get\_abstract\_layer\_info(self, abstract\_api\_name: str) -> dict

获取对应abstract\_api\_name的layer\_info

# TODO 现在仅在MODE1中有临时表

get\_implicit\_para\_name(self, library: str, abstract\_api\_name: str, abstract\_para\_name: str)-> str

获取对应的(abstract\_api\_name, abstract\_para\_name)在library下的implicit\_para\_name

# TODO 现在仅在MODE1中有临时表

get\_seed(self, seed\_name: str) -> dict

获取种子模型，从文件中读取，以字典形式返回。

**配置：**

set\_threshold(self, threshold: float) -> None

赋予该数据库新threshold，并且刷新随threshold刷新的表

get\_threshold(self) -> float

查看当前threshold

refresh\_config(self) -> None

刷新配置，重新读取配置文件，并且数据库中的表基于配置变化做出更新

# Mutate：

Mutate组件只有一个要求，就是输入一个层字典，格式如下：

{

layer: “conv2d”

params: {

in\_channels: 1

out\_channels: 5

kernel\_size: 3

}

in: x

out: x

}

然后，输出一个变异后的层字典（也是抽象的），该变异基于layer\_info中的约束进行，包括api替换和参数替换。

特别的，如果输入的是子模型，那就进行模型变异。

如果进行的是模型变异，返回结构如下：

{

new\_child\_model\_name: [[parameters...], {layer dict}, {layer dict}, ......]

}

接口为：

mutate(layer\_dict: dict) -> (dict, str):

# 随机选择api替换和参数替换，然后返回变异后的新layer\_dict以及变异信息（用于剪枝）

# 注意，此处传入的都是抽象层面的layer\_dict

# ConcretePerformer:

Concrete为主过程，加载对应框架的Performer。

输入一个字典型的模型，将其组装为文件，并返回该模型的各种结果！

concrete会持有一个experiment\_id，调用concrete.new\_experiment()来开始一个新实验，并分配一个新的experiment\_id

.py文件和抽象模型文件都会保存在result / experiment + experiment\_id / seed\_name – gen – index中

结果传递规则：

格式：

{

run test<bool>: 模型运行是否成功。

train test<bool>: 模型训练是否成功

train time cost<float>: 模型训练花费时间

run time cost<float>: 模型运行花费时间

# calculate result<float>: 模型对特定输入的执行结果

case path<str>: 模型路径

# shape result<list[int]>: 模型对特定输入的输出形状

error message<str>: 报错信息，如果没有报错则为””

}

在主过程中，该字典会加入mutate info

接口为：

perform(abstract\_model: dict, gen: int, index: int) -> list[dict]

# ResultAnalyse:

分析ConcretePerformer传回来的结果，记录在指定的目录中，并且：为主过程返回True或False以判定该模型是否还能继续向下变异。

接口为：

analyse\_result(result\_list: list[dict]) -> bool

# 主过程：

# TODO