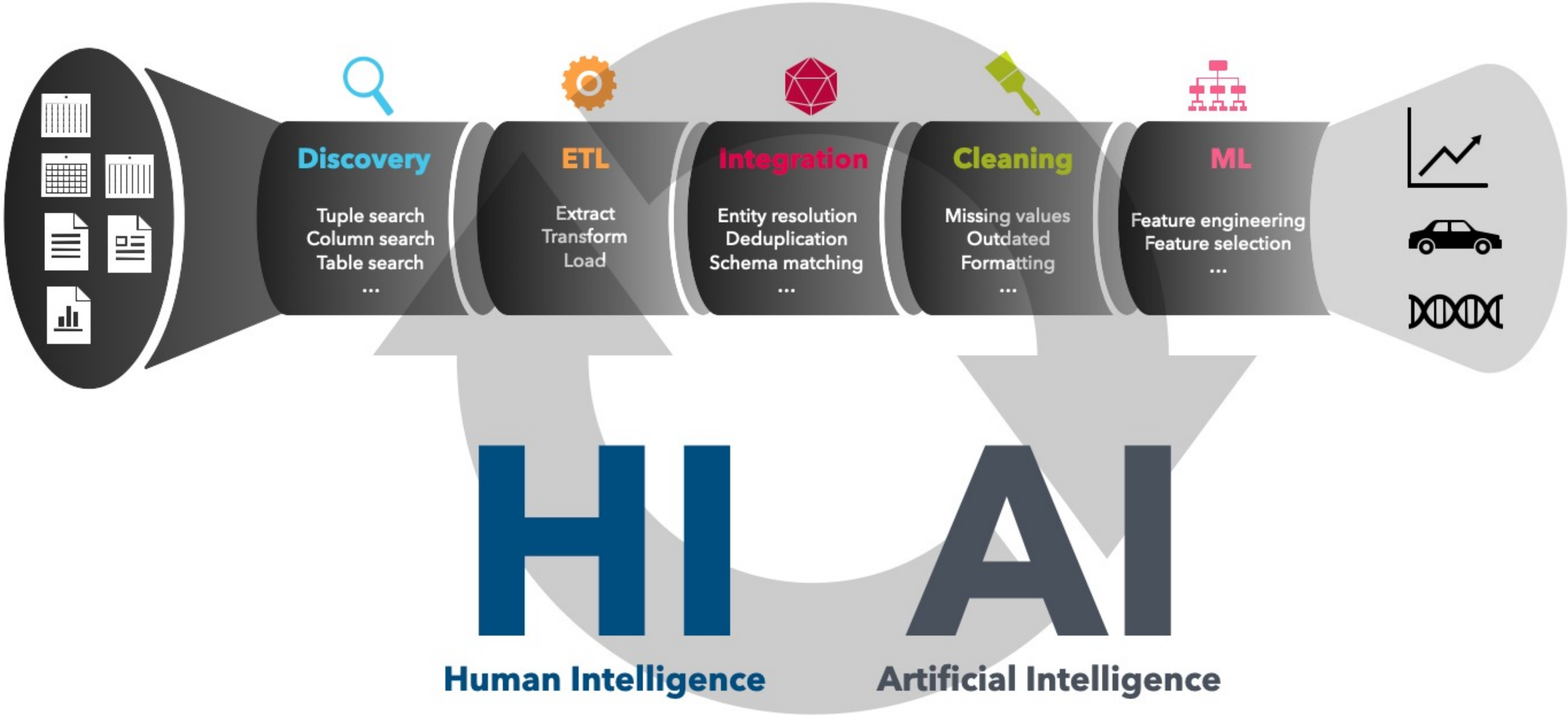
Orchestrating Data Preparation Pipelines

Orchestrating Data Preparation Pipelines



Outline

- Overview
  - Motivation
  - Challenges
  - Manual Pipeline Orchestration
  - Automatic Pipeline Generation
  - Human-in-the-loop Pipeline Generation
- Open Problems

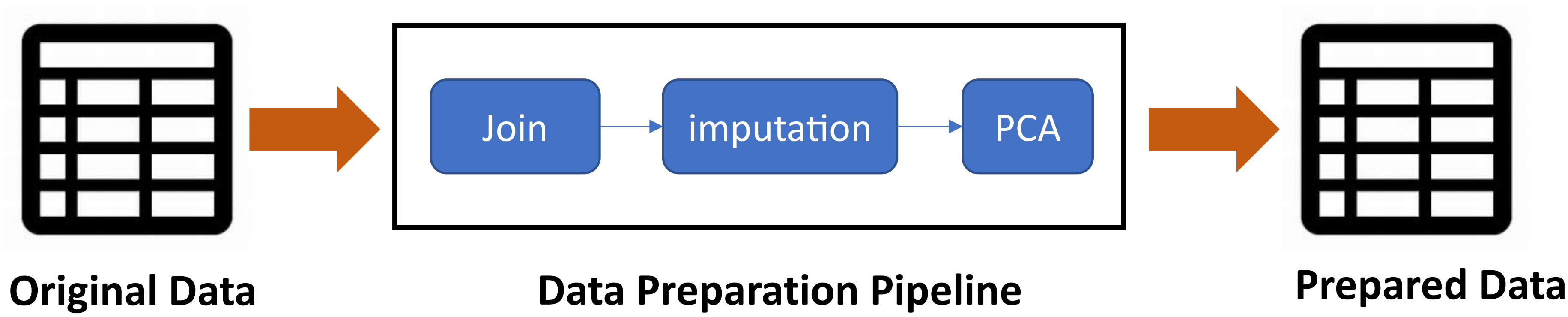
Outline

- Overview
  - Motivation
  - Challenges
  - Manual Pipeline Orchestration
  - Automatic Pipeline Generation
  - Human-in-the-loop Pipeline Generation
- Open Problems

Motivation

□ Data Preparation Pipeline

- ➢ Requires a series steps
  - ➢ data wrangling, data cleaning, feature engineering…



□ Limitations:

- ➢ Rely on experts
- ➢ Time-consuming
- ➢ Hard to discover the optimal solution

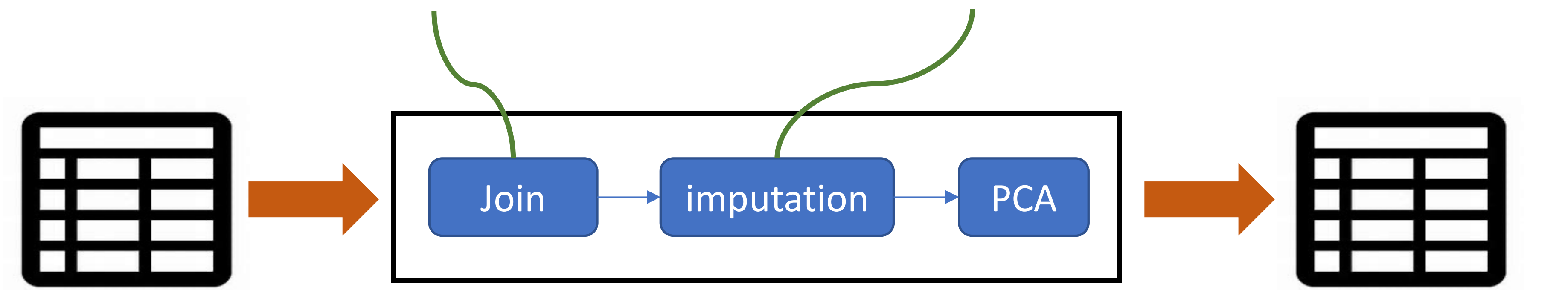
Challenges

☐Large and complex search space

- ➢Each step can be implemented by different algorithms
- ➢Complex dependencies among operators

- Merge Join
- Inner Join
- Nested Loops Join
- Hash Join

- Mean and Median
- Dropping
- Statistics-based
- ML-based

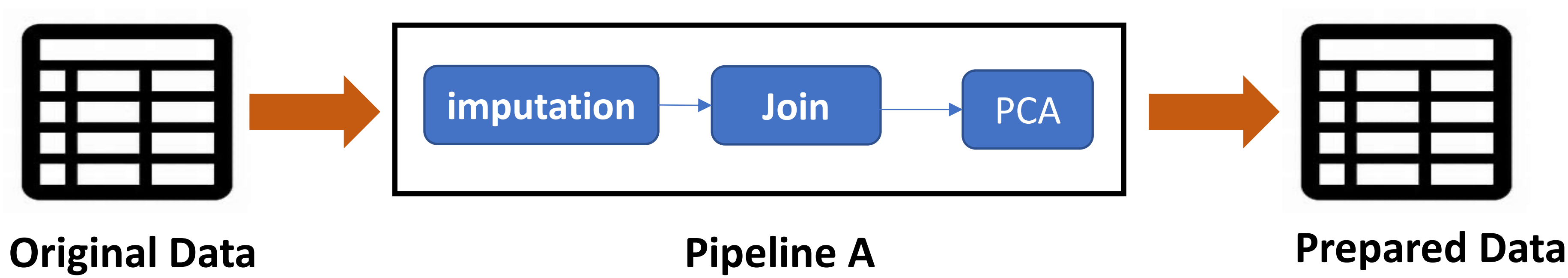


Original Data **Data Preparation Pipeline** **Prepared Data**

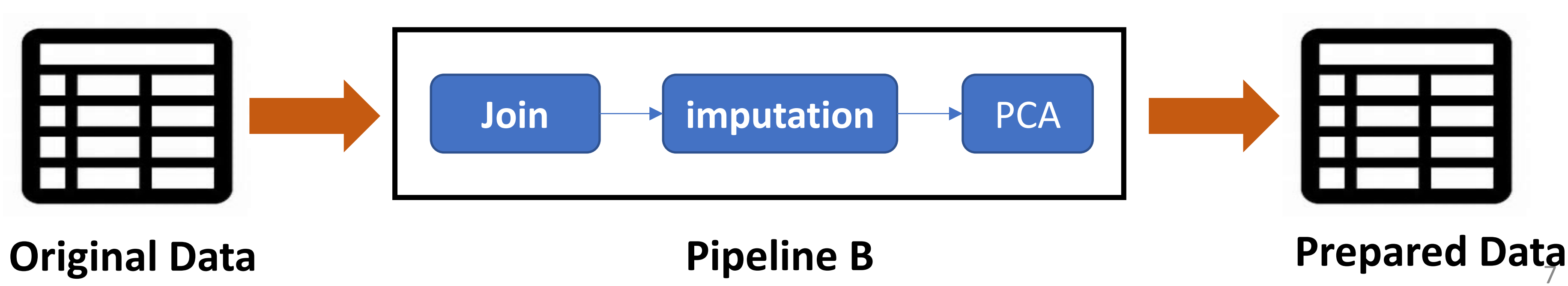
Challenges

☐ Domain- or even dataset-specific

- ➢ Dependency of downstream tasks
- ➢ Dependency of underlying datasets



Which one is better?

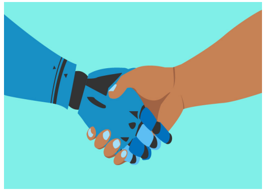


Three Types of Data Preparation Pipelines

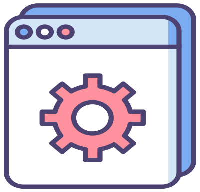
Expensive

Human Effort

Manual Pipeline Orchestration



Human-in-the-loop Pipeline Generation



Automatic Pipeline Generation

Cheap

Outline

- **Overview**
  - **Motivation**
  - **Challenges**
  - **Manual Pipeline Orchestration**
  - Automatic Pipeline Generation
  - Human-in-the-loop Pipeline Generation
- Open Problems

An Example

◻Hand-written script

- UDFs
- Domain Knowledges

```python
df = pd.read_csv("KaggleV2-May-2016.csv")

df = df[(df.Age >= 0) & (df.Age <= 100)]

df['ScheduledDay'] = pd.to_datetime(df['ScheduledDay'])
df['AppointmentDay'] = pd.to_datetime(df['AppointmentDay'])
df['AwaitingTime'] = df["AppointmentDay"].sub(df["ScheduledDay"], axis=0)
df["AwaitingTime"] = (df["AwaitingTime"] / np.timedelta64(1, 'D')).abs()

df.drop(['PatientId', 'AppointmentID', 'ScheduledDay',
        'Handcap', 'AppointmentDay', 'Neighbourhood'
        ], axis=1, inplace=True)

X = df.drop("No-show",axis=1)
y = df["No-show"]

X_train1 = pd.get_dummies(X)
y.replace("No", 0,inplace=True)
y.replace("Yes", 1,inplace=True)

scaler = StandardScaler().fit(X_train1)
rescaledX2 = scaler.transform(X_train1)

X_train, X_test, y_train, y_test = train_test_split(
    rescaledX2, y, train_size=0.8, test_size=1-0.8, random_state=0)
```

1. Dealing Outlier

2. Feature Augmentation

3. Removing Irrelevance Features

4.Encoding

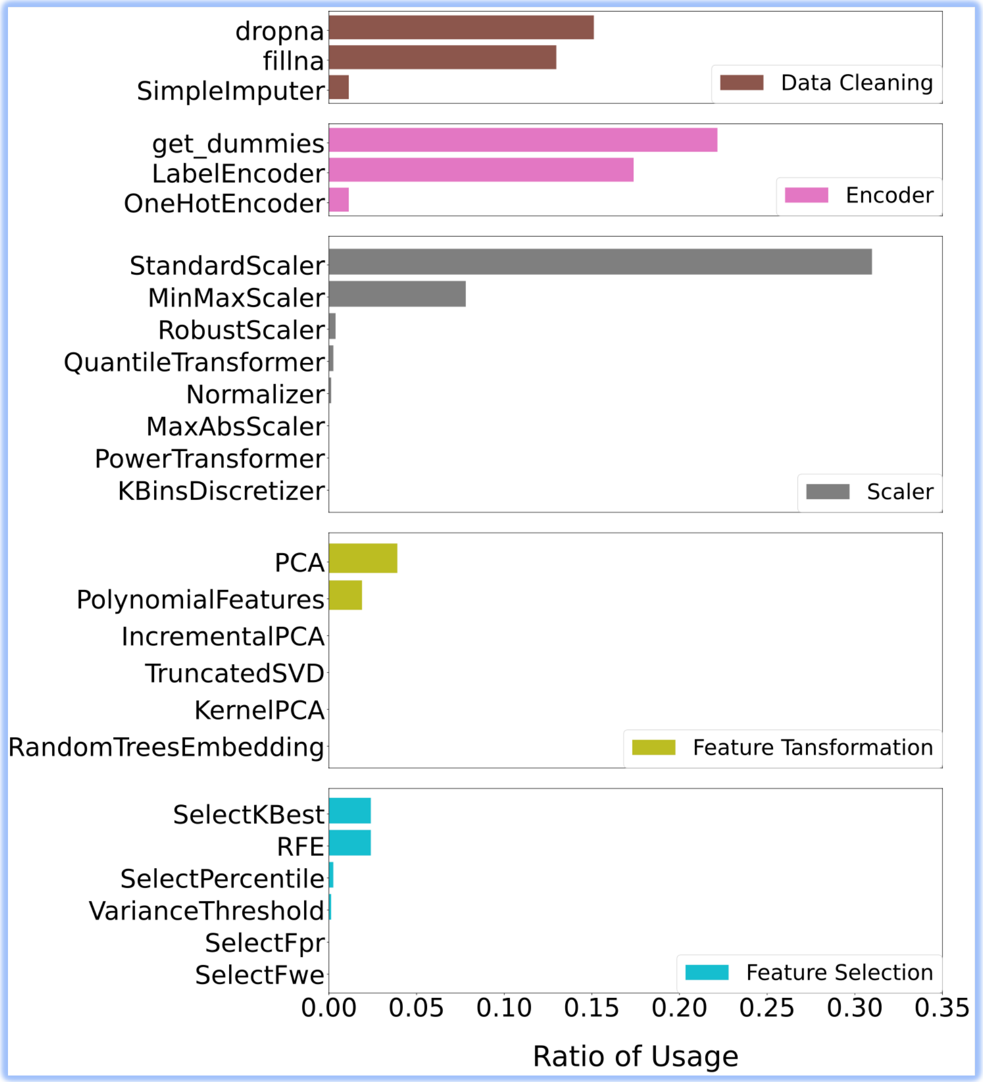
5.Scaling

6. Train-Test Splitting

Manual Pipeline Analysis

☐Operator Level

- ➢ **Data Cleaning**
- ➢ **Encoder**
- ➢ **Scaler**
- ➢ **Feature Transformation**
- ➢ **Feature Selection**

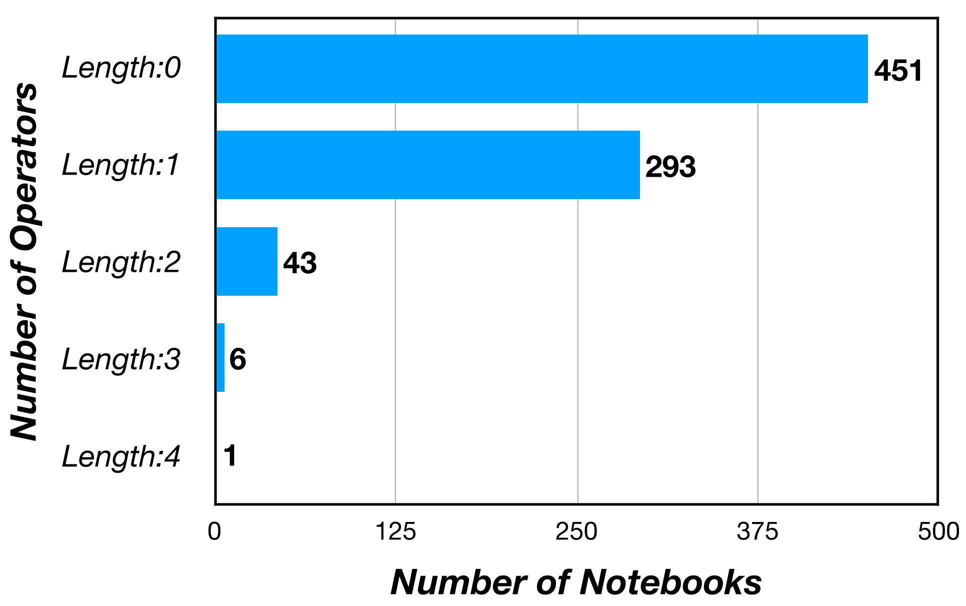


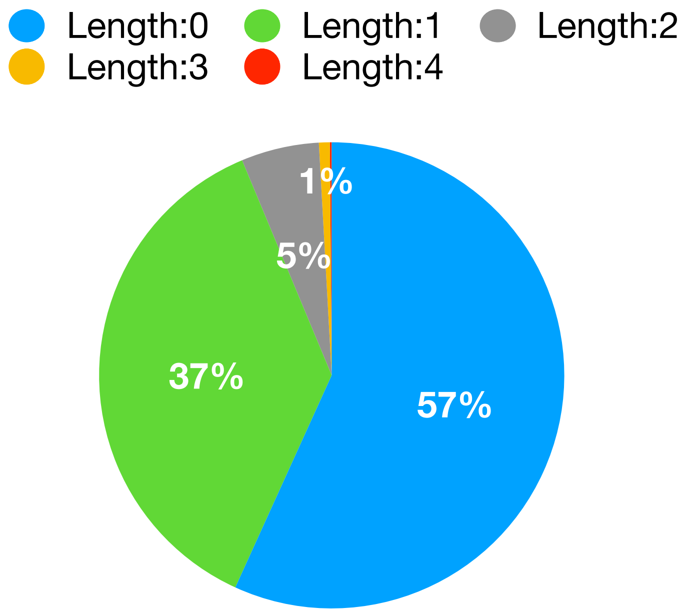
Analysis based on 800 notebooks from Kaggle

Manual Pipeline Analysis

☐Pipeline Level

- ➢ #-Operators vs. #-Notebooks

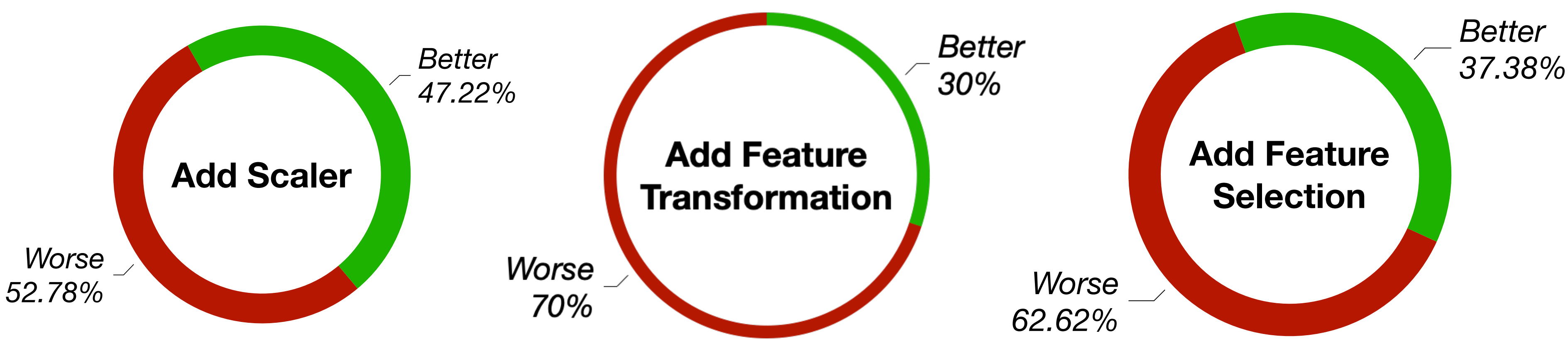




Manual Pipeline Analysis

☐Pipeline Level

- Performance after adding operators



Take-away

Pros

- These pipelines are very flexible.
- These pipelines can be easily injected with domain knowledge and user experiences.

Cons

- Human orchestrated pipelines may have “blind spots”.

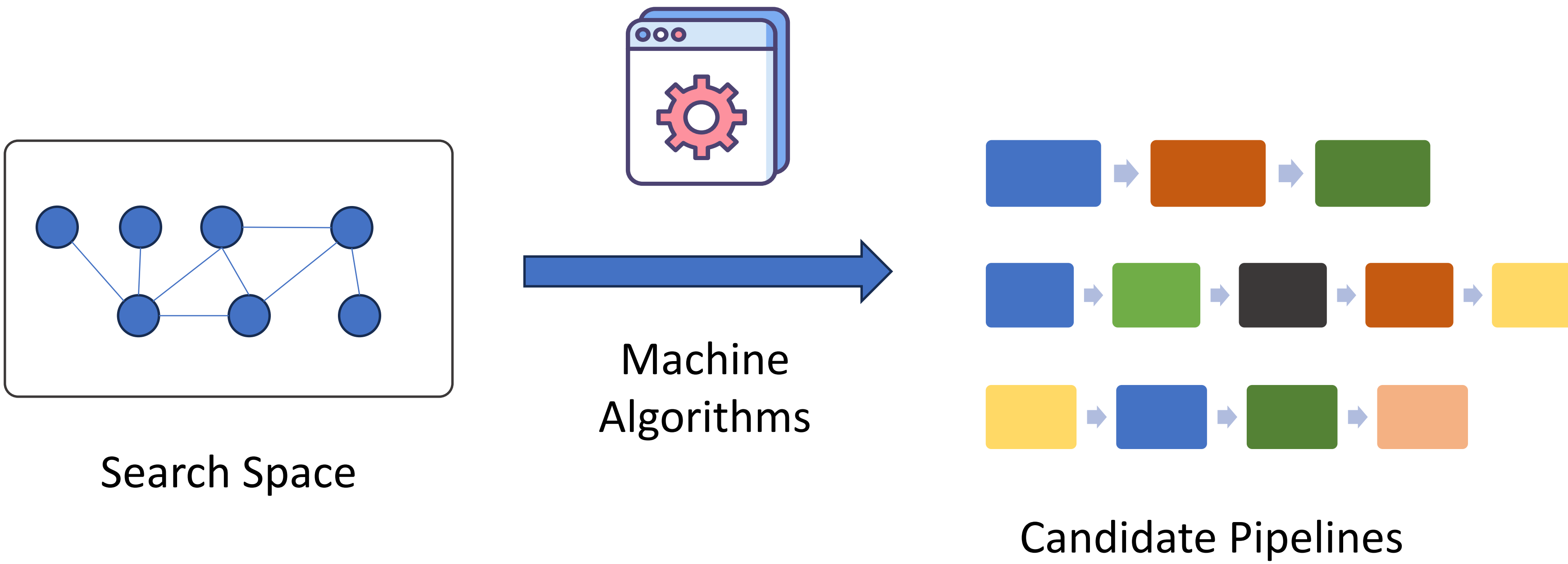
Can we automatically generate the pipeline?

- Reduce human effort ↓
- Improve the performance ↑

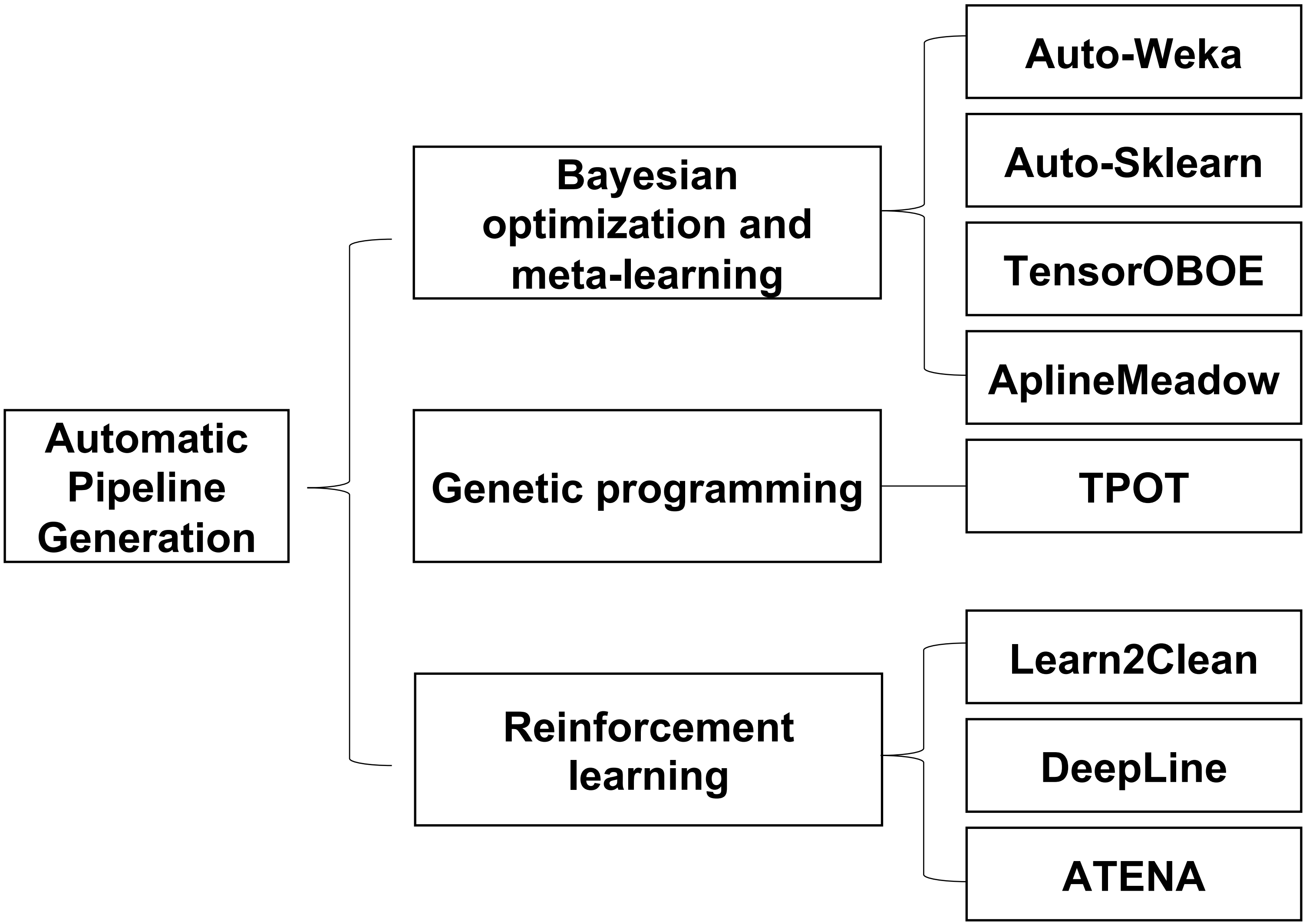
Outline

- **Overview**
  - **Motivation**
  - **Challenges**
  - **Manual Pipeline Orchestration**
  - **Automatic Pipeline Generation**
  - Human-in-the-loop Pipeline Generation
- Open Problems

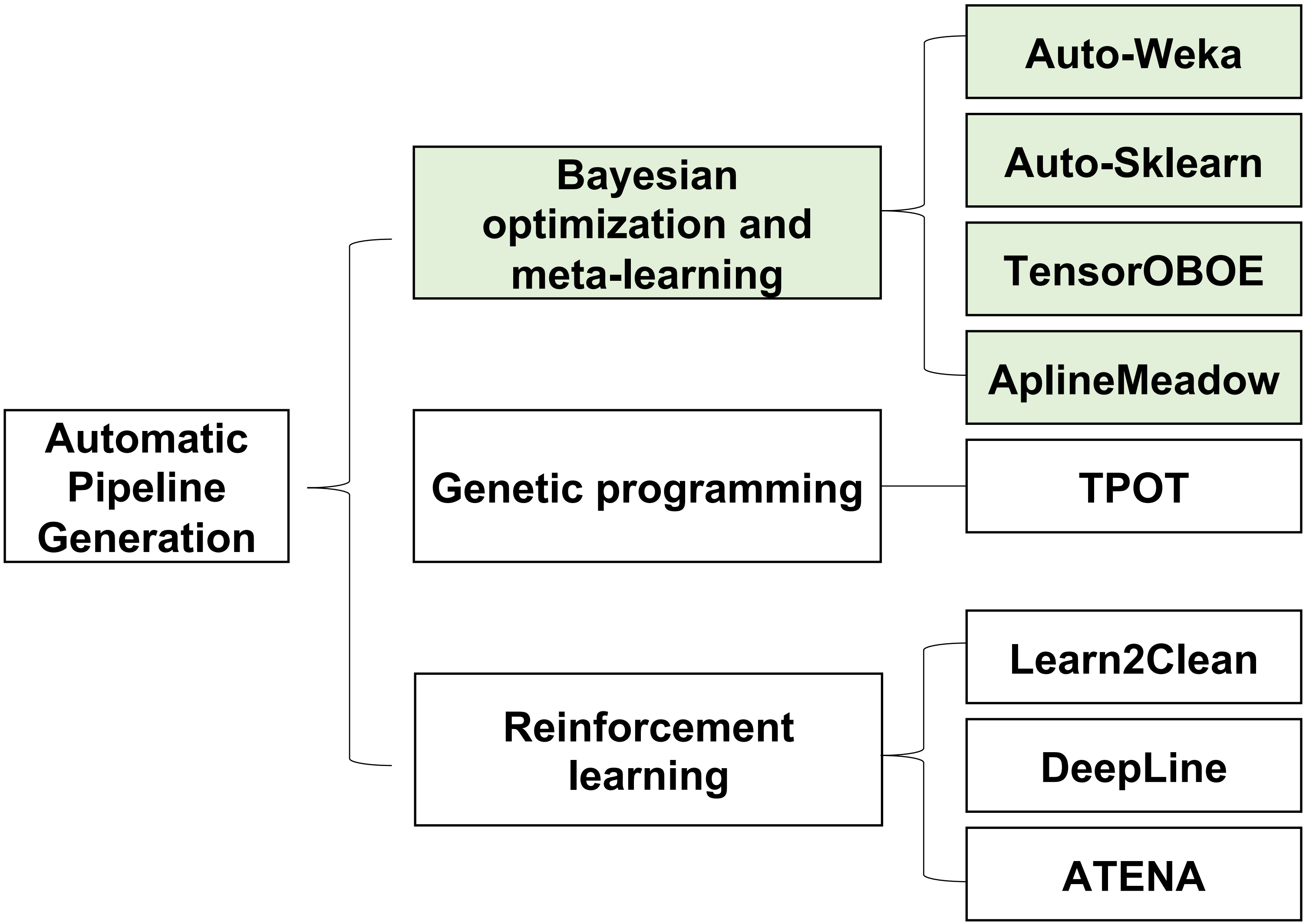
Automatic Pipeline Generation



Automatic Pipeline Generation



Automatic Pipeline Generation



Auto-WEKA

☐ Problem definition:

- CASH: Combined Algorithm Selection and Hyperparameter optimization

☐ Key Idea:

- Bayesian optimization
- p(c | λ)

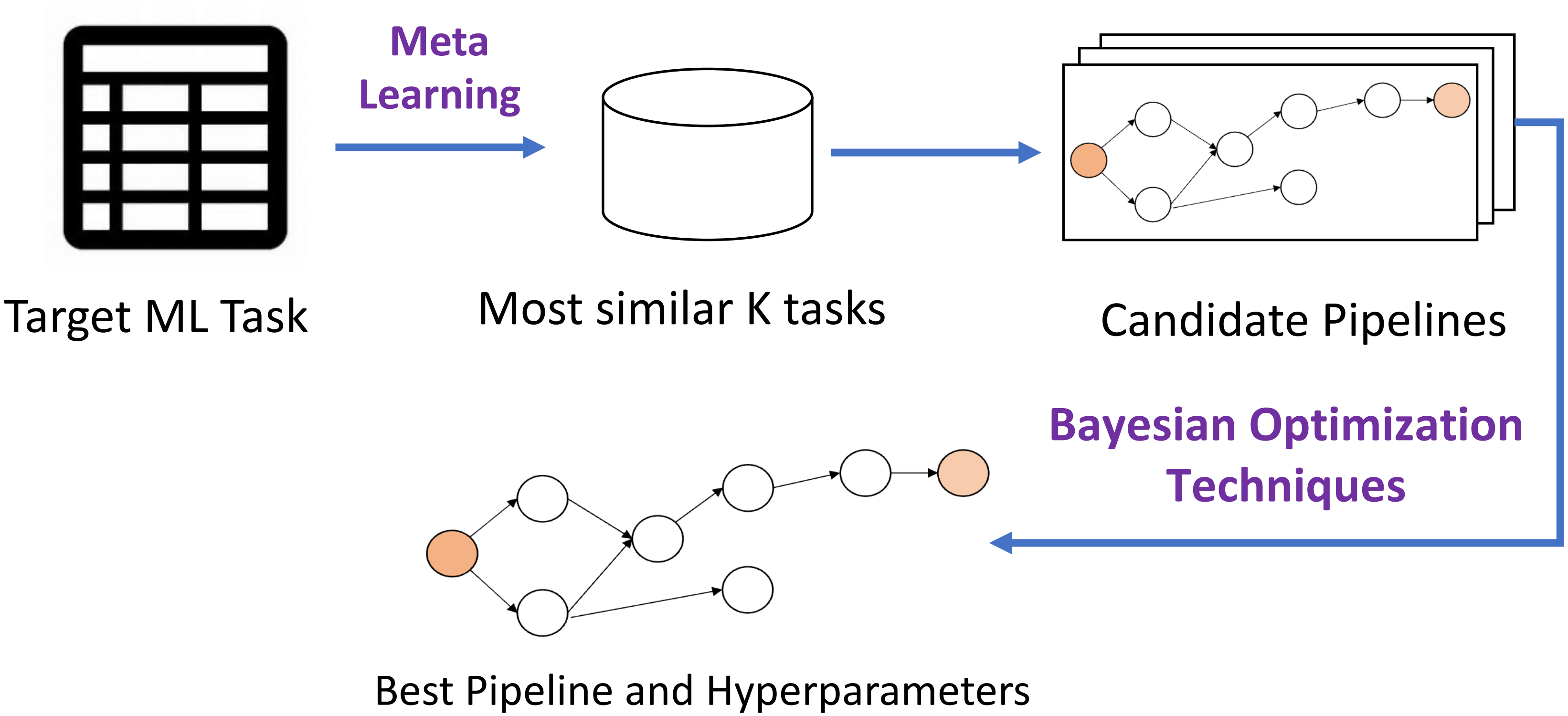
Algorithm 1 SMBO

initialise model $\mathcal{M}_L$; $\mathcal{H} \leftarrow \emptyset$
while time budget for optimization has not been exhausted **do**
  $\boldsymbol{\lambda} \leftarrow$ candidate configuration from $\mathcal{M}_L$
  Compute $c = \mathcal{L}(A_{\boldsymbol{\lambda}}, \mathcal{D}^{(i)}_{\text{train}}, \mathcal{D}^{(i)}_{\text{valid}})$
  $\mathcal{H} \leftarrow \mathcal{H} \cup \{(\boldsymbol{\lambda}, c)\}$
  Update $\mathcal{M}_L$ given $\mathcal{H}$
end while
return $\boldsymbol{\lambda}$ from $\mathcal{H}$ with minimal c

[1] Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms

Auto-Sklearn

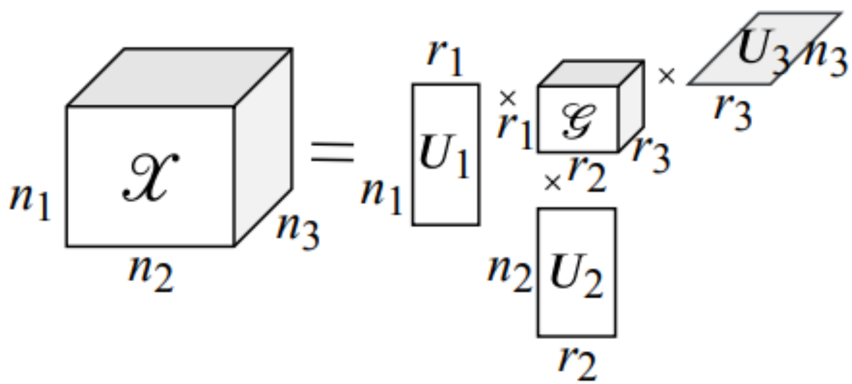
- **Key Idea**
  - Meta Learning for coarse-grained pipeline selection
  - Bayesian Optimization for fine-grained pipeline generation



[2] Matthias Feurer et al. Efficient and Robust Automated Machine Learning. NIPS 2015.

TensorOBOE

- **TensorOBOE:** a new structured model based on tensor decomposition for AutoML pipeline selection
- **Key Idea**
  - Use low rank tensor decomposition as a surrogate model for efficient pipeline search
  - Use meta-learning to optimize an error matrix, which can be decomposed as 6 matrices

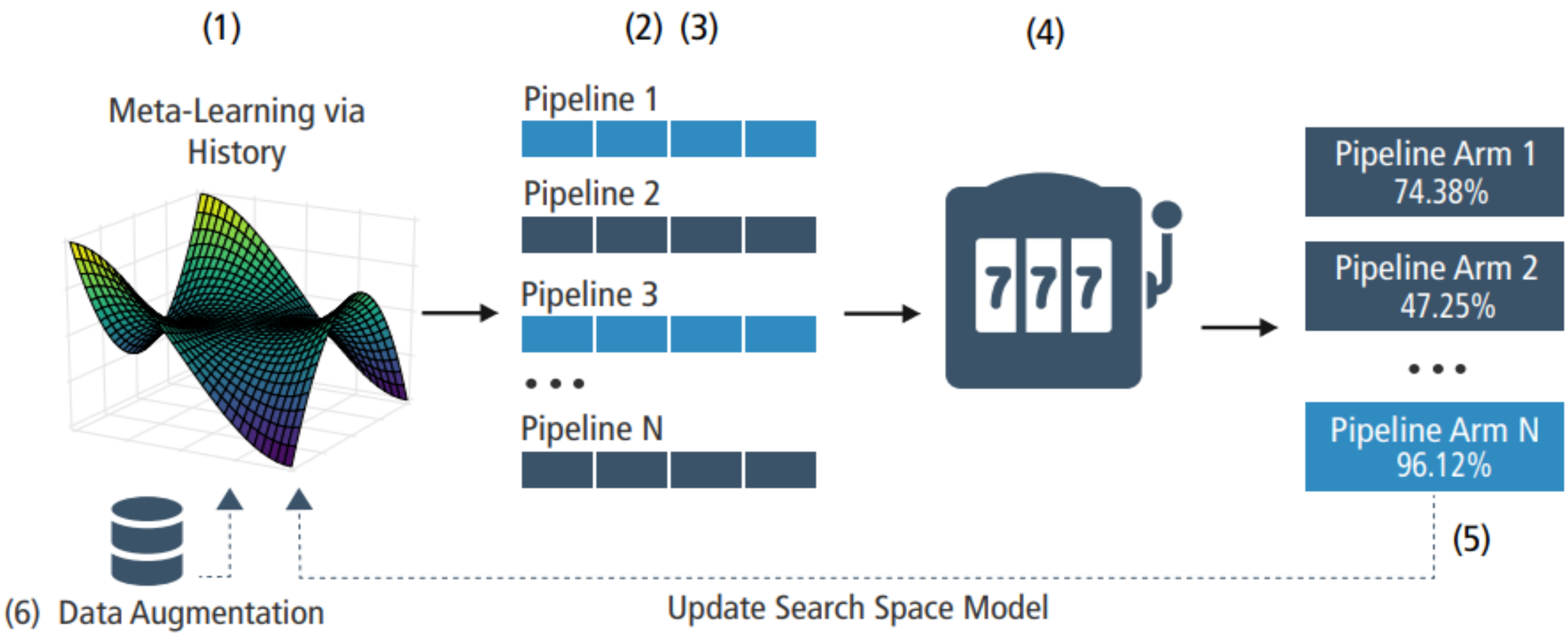


[3] C. Yang el al. AutoML Pipeline Selection: Efficiently Navigating the Combinatorial Space. SIGKDD 2020

Alpine Meadow

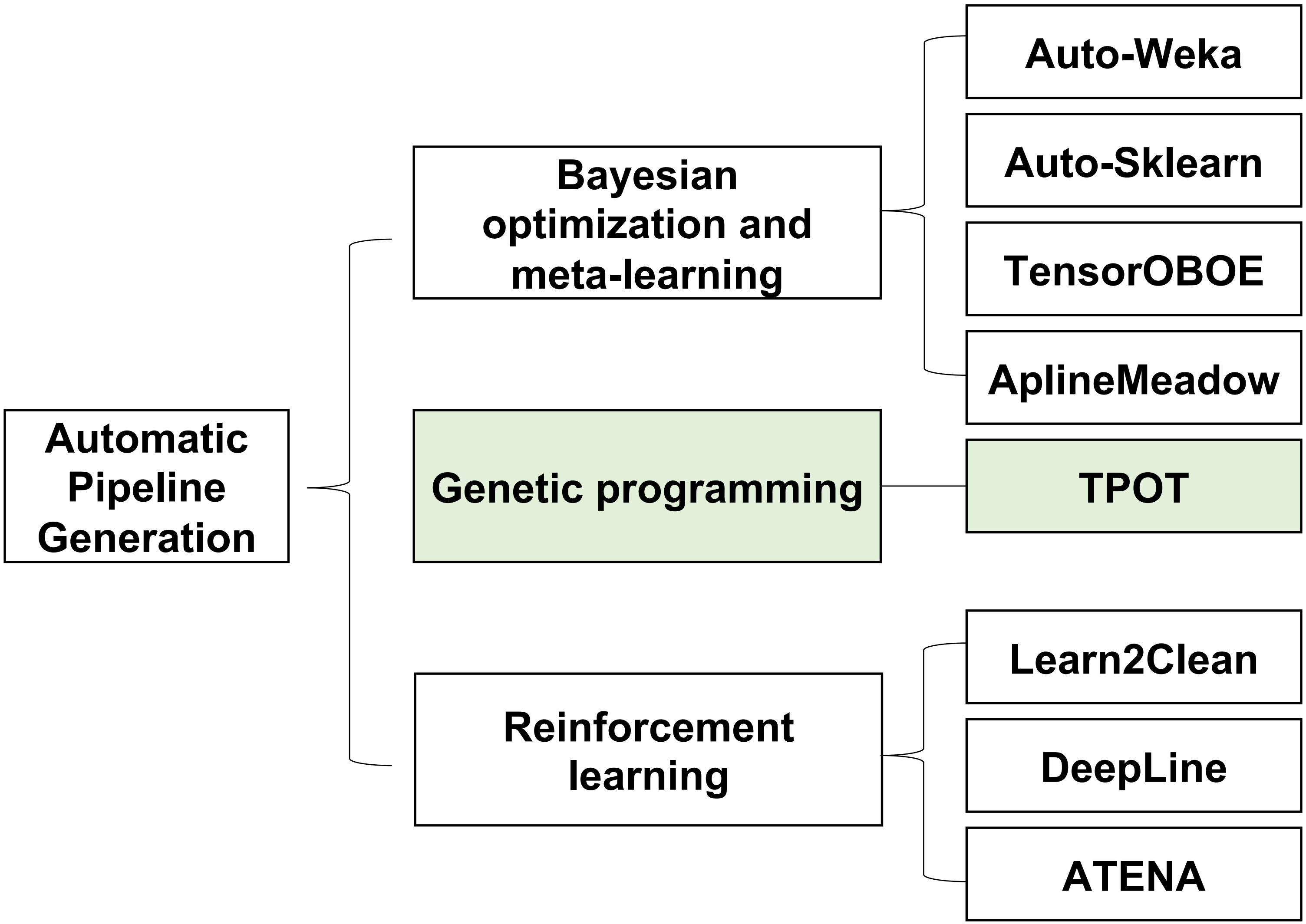
☐ Key Idea

- Rule-based optimization, can be combined with multi-armed bandits, Bayesian optimization and meta-learning



[4] Z. Shang. et al.Democratizing Data Science through Interactive Curation of ML Pipelines. SIGMOD 2019

Automatic Pipeline Generation



TPOT

- **Key Idea**
  - A tree-based representation model of data preparation pipelines
  - optimize the pipelines using genetic programming

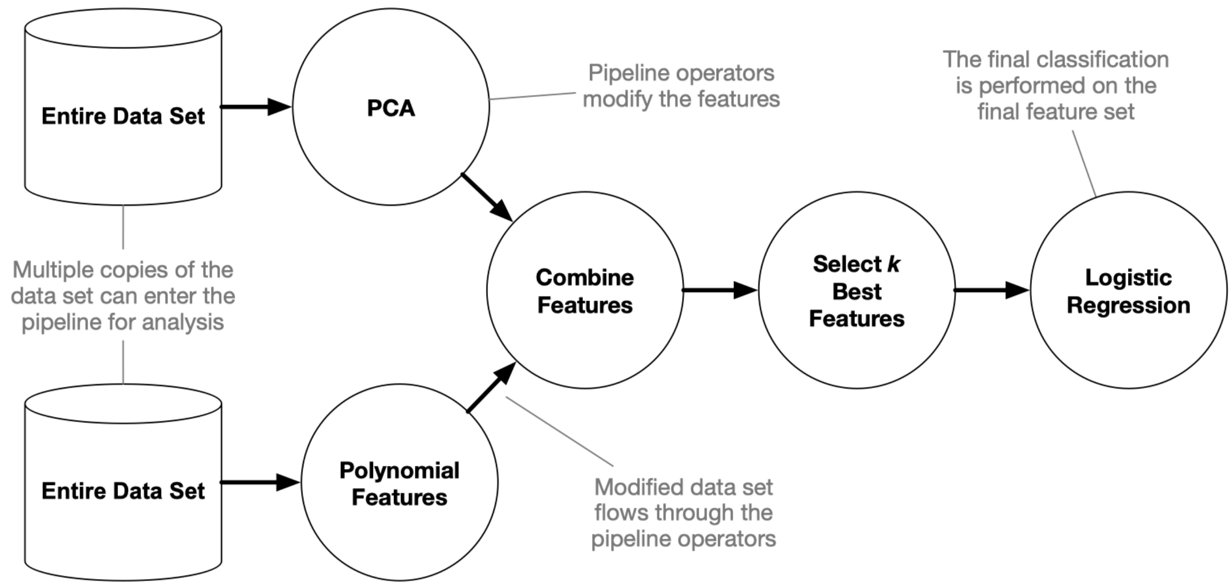


Figure 1: An example tree-based pipeline from TPOT. Each circle corresponds to a machine learning operator, and the arrows indicate the direction of the data flow.

[5] R. S. Olson el al. TPOT: A tree-based pipeline optimization tool for automating machine learning. AutoML@ICML 2016

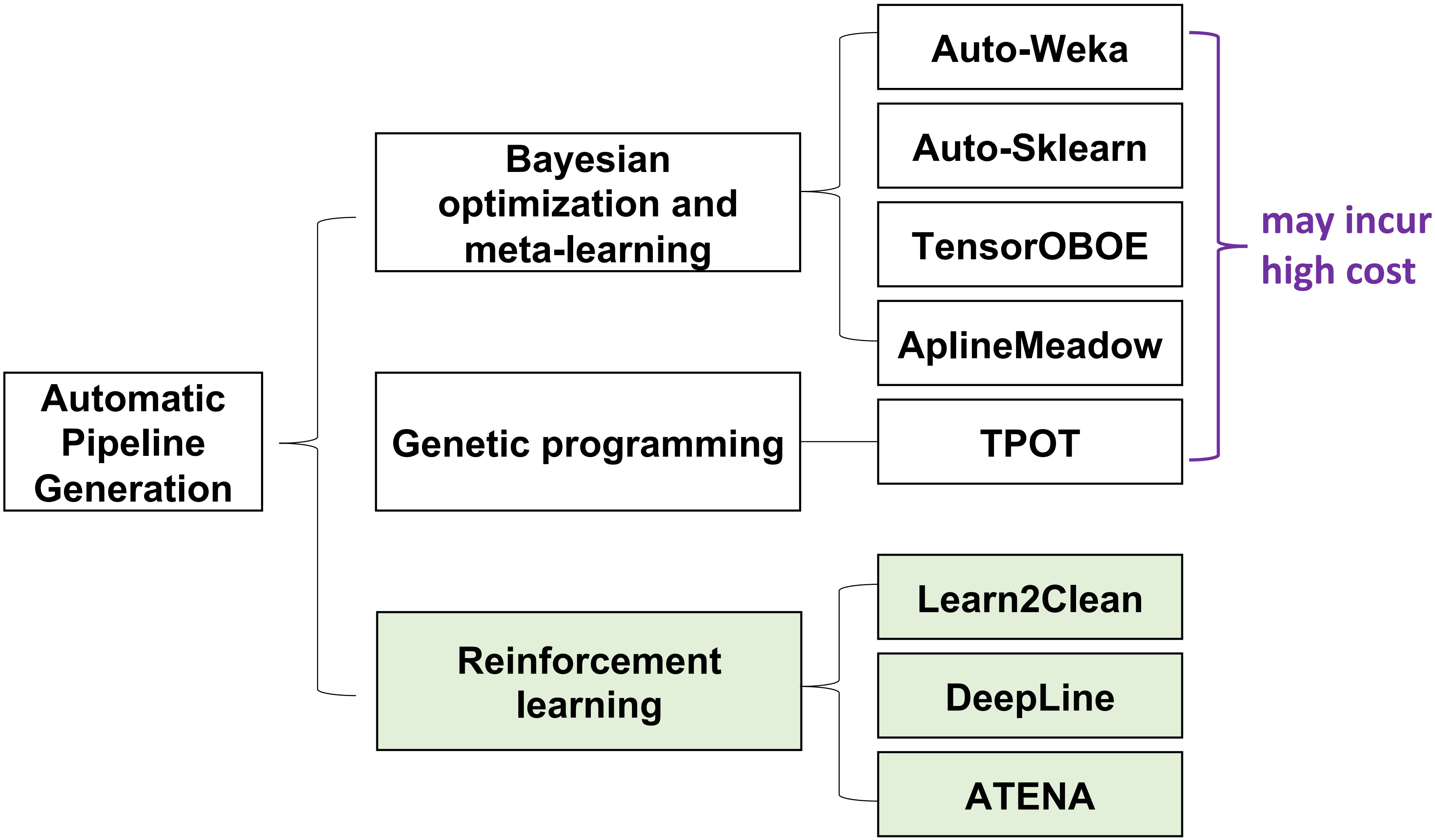
TPOT

- **Key Idea**
  - A tree-based representation model of data preparation pipelines
  - optimize the pipelines using genetic programming

- **Key Steps**
  - Step1: Random generate 100 pipelines.
  - Step2: Select 20 best pipelines.
  - Step3: Each of the top 20 selected pipelines produce five copies (i.e., offspring) into the next generation's population
  - Step4: Repeat this evaluate-select-crossover-mutate process for 100 generations.

[5] R. S. Olson el al. TPOT: A tree-based pipeline optimization tool for automating machine learning. AutoML@ICML 2016

Automatic Pipeline Generation



Reinforcement Learning

☐Key Idea

- Model Data Preparation as the Markov Decision Process
- RL predicts data preparation operator step-by-step

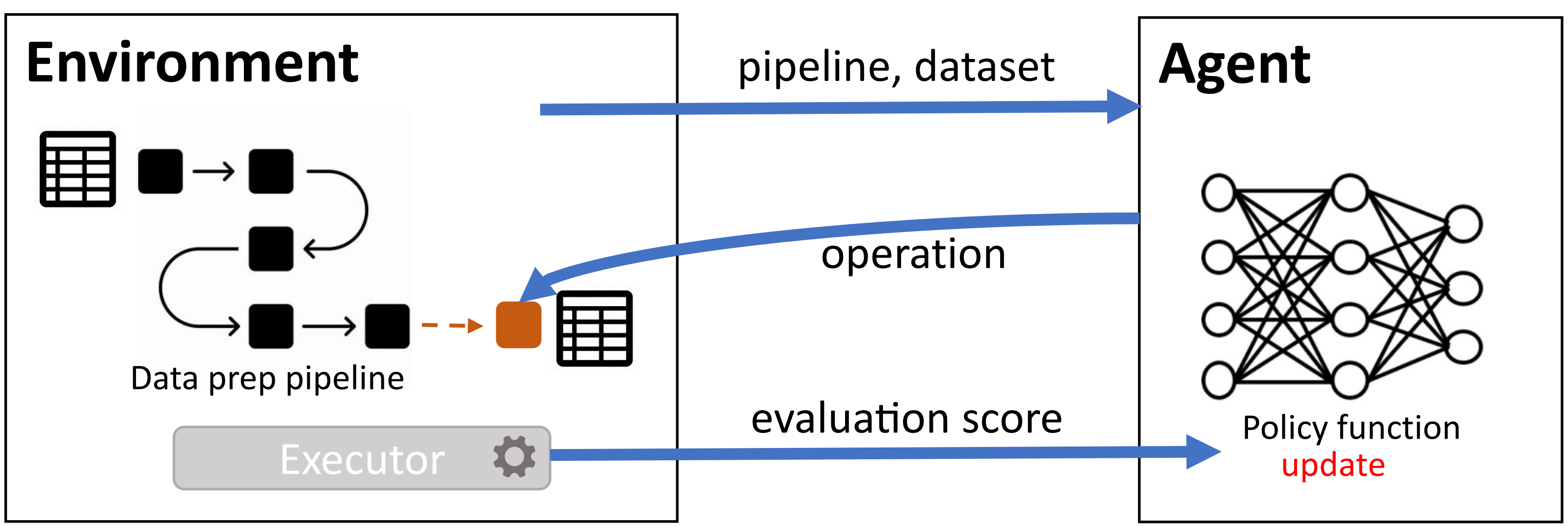
☐Goal

- The data prepared through this series of operations can achieve the best results in machine learning tasks

Reinforcement Learning

☐ General Framework

- **State:** vector of dataset and pipeline;
- **Action:** a set of data preparation operations;
- **Reward:** ML evaluation result.
- **Transition function:** add an action (operation) to pipeline and execute it to generate a new dataset.



Learn2Clean

☐ Aim at orchestrating data cleaning pipeline

- ➢ Decision strategy is optimized by Q-Learning

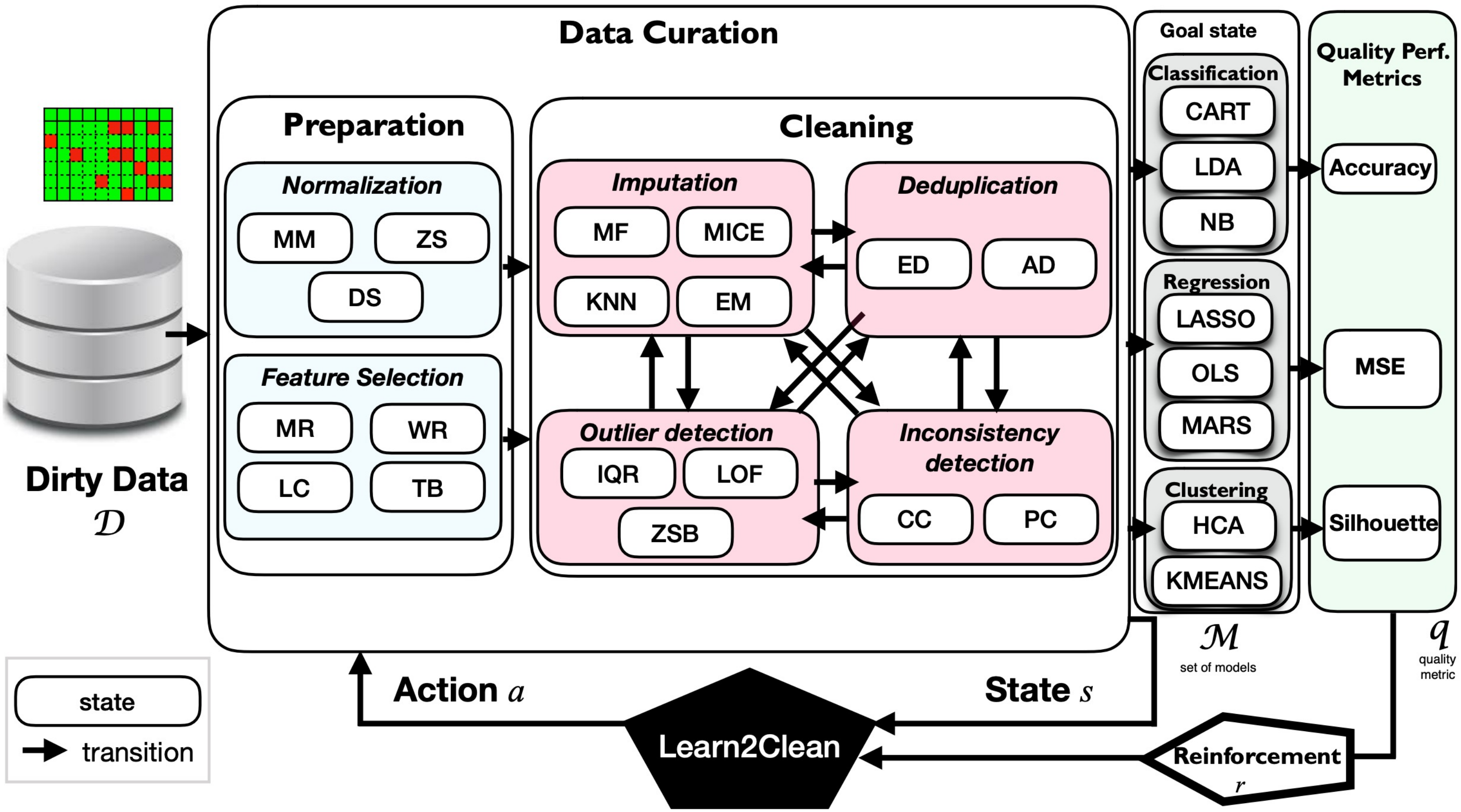


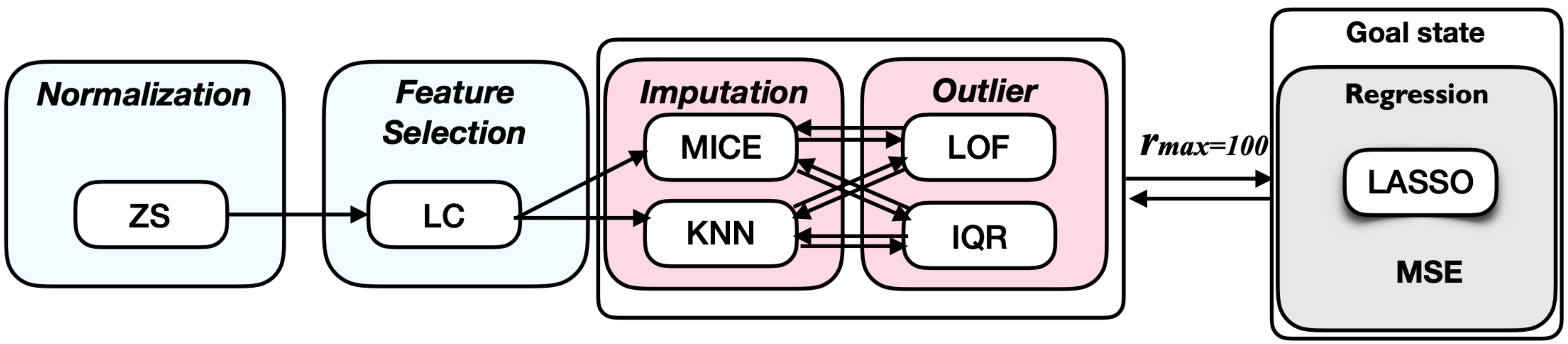
Figure 1: Learn2Clean Architecture

[5] Laure Berti-Equille. Learn2Clean: Optimizing the Sequence of Tasks for Web Data Preparation. WWW 2019

Learn2Clean

Key Idea

- Decision strategy is optimized by Q-Learning
- Learn2Clean uses a Q-value matrix to model the value of selection for each state



$$R_{init} = \begin{array}{c c} & \textbf{actions} \\ \textbf{state} & \begin{array}{ccccccc} \text{ZS} & \text{LC} & \text{MICE} & \text{KNN} & \text{LOF} & \text{IQR} & \text{LASSO} \end{array} \\ \begin{array}{c} \text{ZS} \\ \text{LC} \\ \text{MICE} \\ \text{KNN} \\ \text{LOF} \\ \text{IQR} \\ \text{LASSO} \end{array} & \left[\begin{array}{ccccccc} -1 & 0 & 0 & 0 & 0 & 0 & -1 \\ -1 & -1 & 0 & 0 & -1 & -1 & -1 \\ 0 & -1 & -1 & -1 & 0 & 0 & 100 \\ 0 & -1 & -1 & -1 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & -1 & -1 & 100 \\ 0 & 0 & 0 & 0 & -1 & -1 & 100 \\ -1 & -1 & 0 & 0 & 0 & 0 & -1 \end{array}\right] \end{array}$$

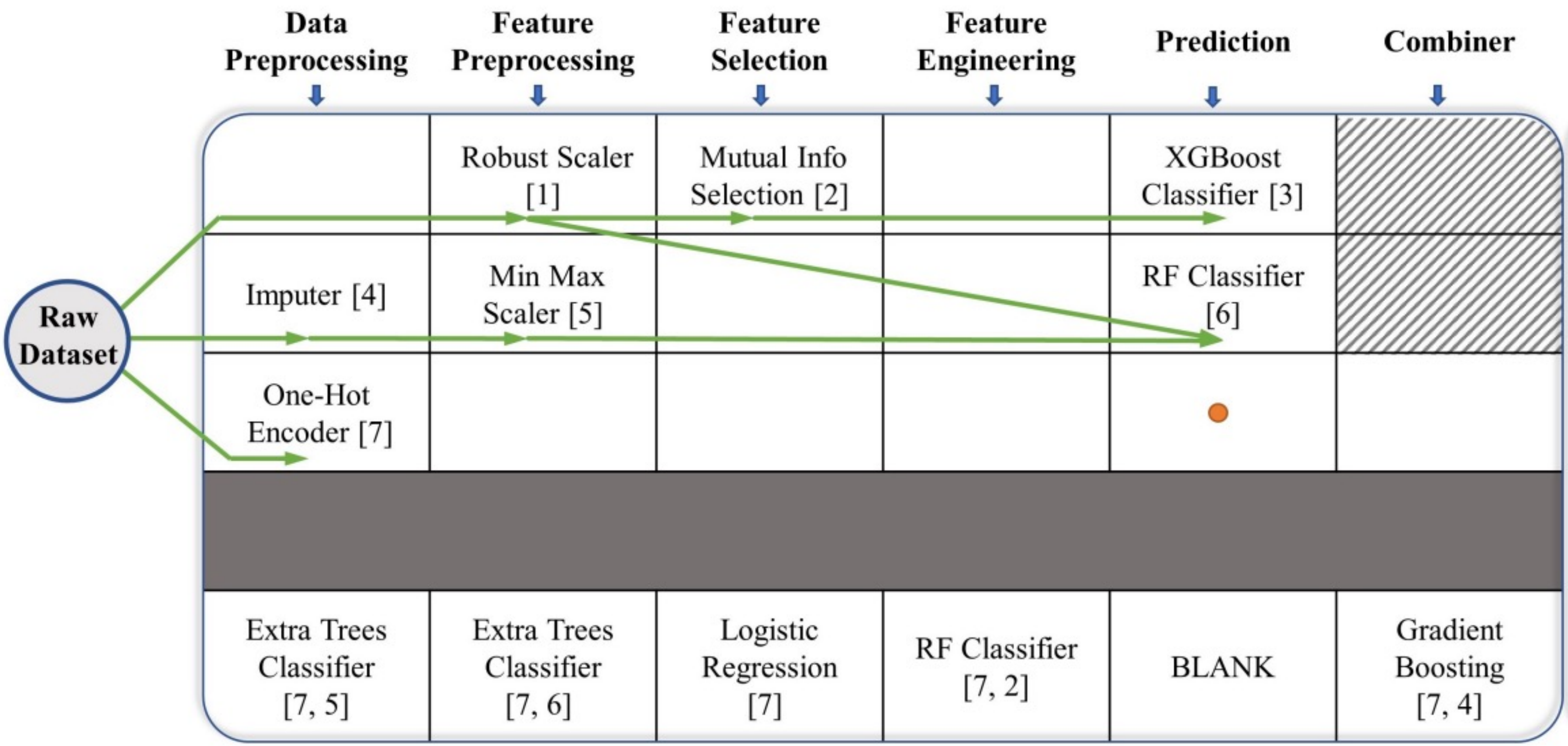
Reward: $r' = \beta(Norm(s, q_m) - Norm(s', q'_m))$

[5] Laure Berti-Equille. Learn2Clean: Optimizing the Sequence of Tasks for Web Data Preparation. WWW 2019

DeepLine

☐ Goal:

- ➢ Automatic generation of end-to-end ML pipelines



Model a pipeline as a grid of operation

[6] Y. Heffetz el al. DeepLine: AutoML Tool for Pipelines Generation using Deep Reinforcement Learning and Hierarchical Actions Filtering.

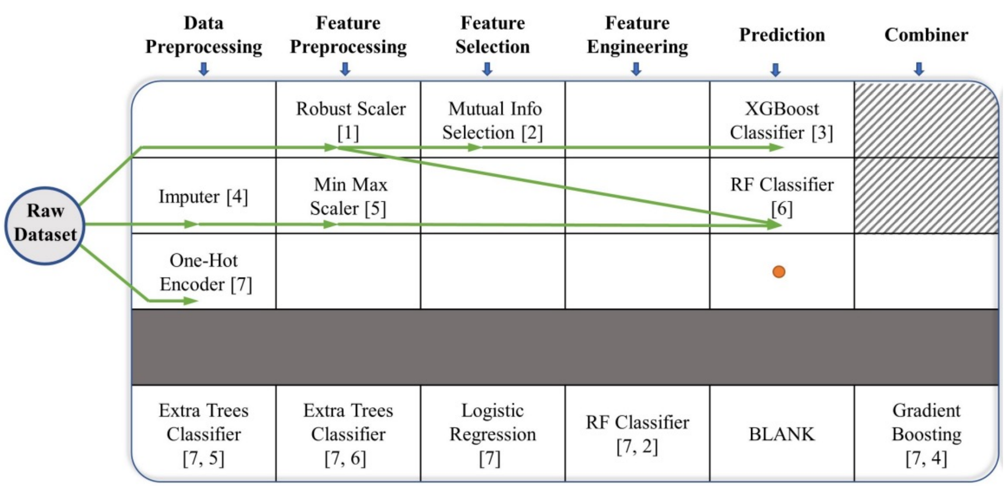
DeepLine

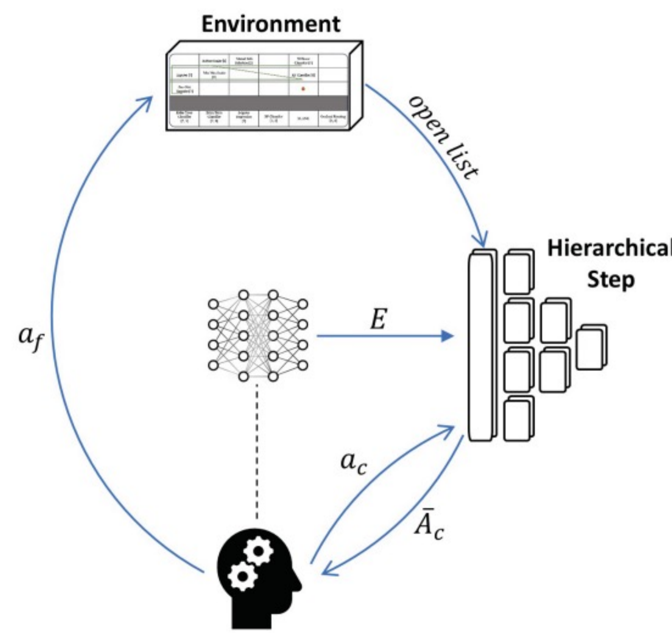
☐ Goal:

- ➢ Automatic generation of end-to-end ML pipelines

☐ Key Idea

- ➢ DeepLine uses DQN to optimize the policy strategy of selecting operation in each node of the grid.
- ➢ Agent: Hierarchical action-modeling approach for modelling dynamic action spaces



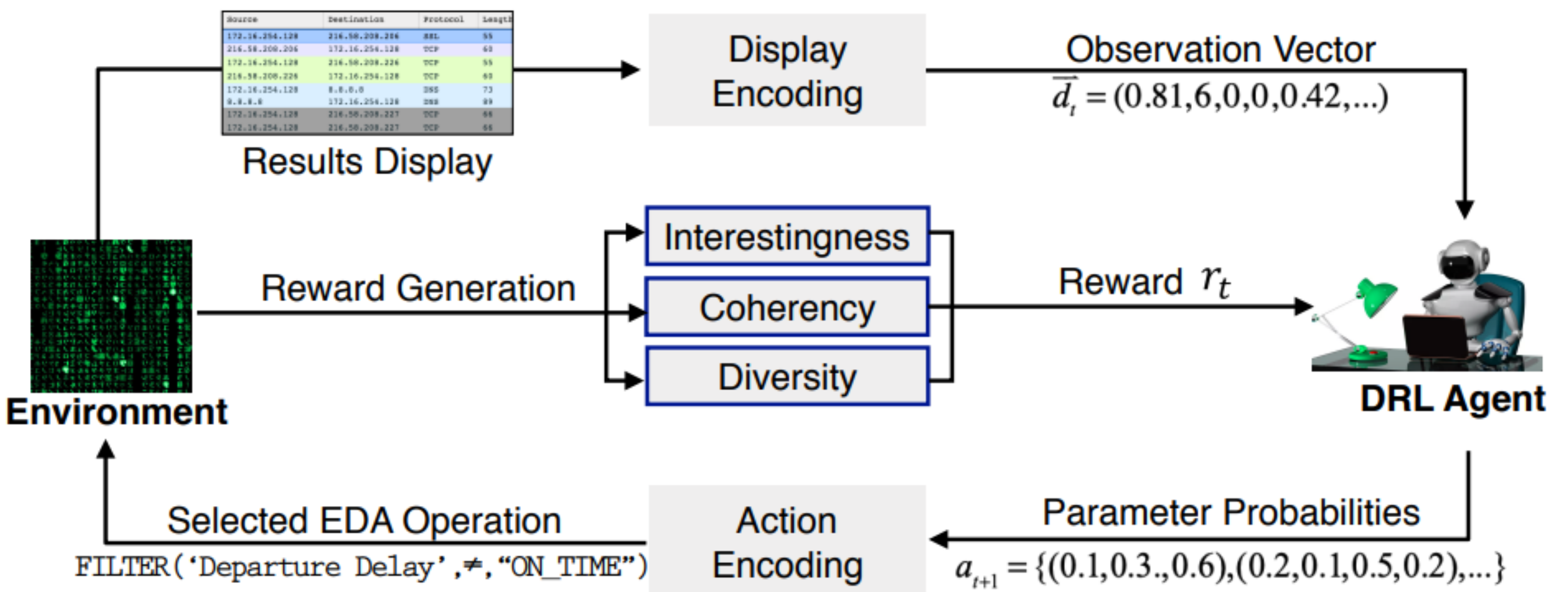


Pipeline -> A Grid of Operation

[6] Y. Heffetz el al. DeepLine: AutoML Tool for Pipelines Generation using Deep Reinforcement Learning and Hierarchical Actions Filtering.

ATENA

- **Goal**
  - Automatically Generating Exploratory Data Analysis (EDA) Pipeline
- **Key Idea**
  - Formulate the EDA process as the Markov Decision Process
  - Deep reinforcement learning with domain-specific reward function



[7] O.El et al. Automatically Generating Data Exploration Sessions Using Deep Reinforcement Learning. SIGMOD 2020.

Take-away

Pros

- Automatic generation, blink and it's done
- Lower the barriers to a good data preparation pipeline

Cons

- May be misled by blindly suggesting possibly good pipeline
- Hard to incorporate the user expertise

Can we involve users into the Auto-pipeline generation process?

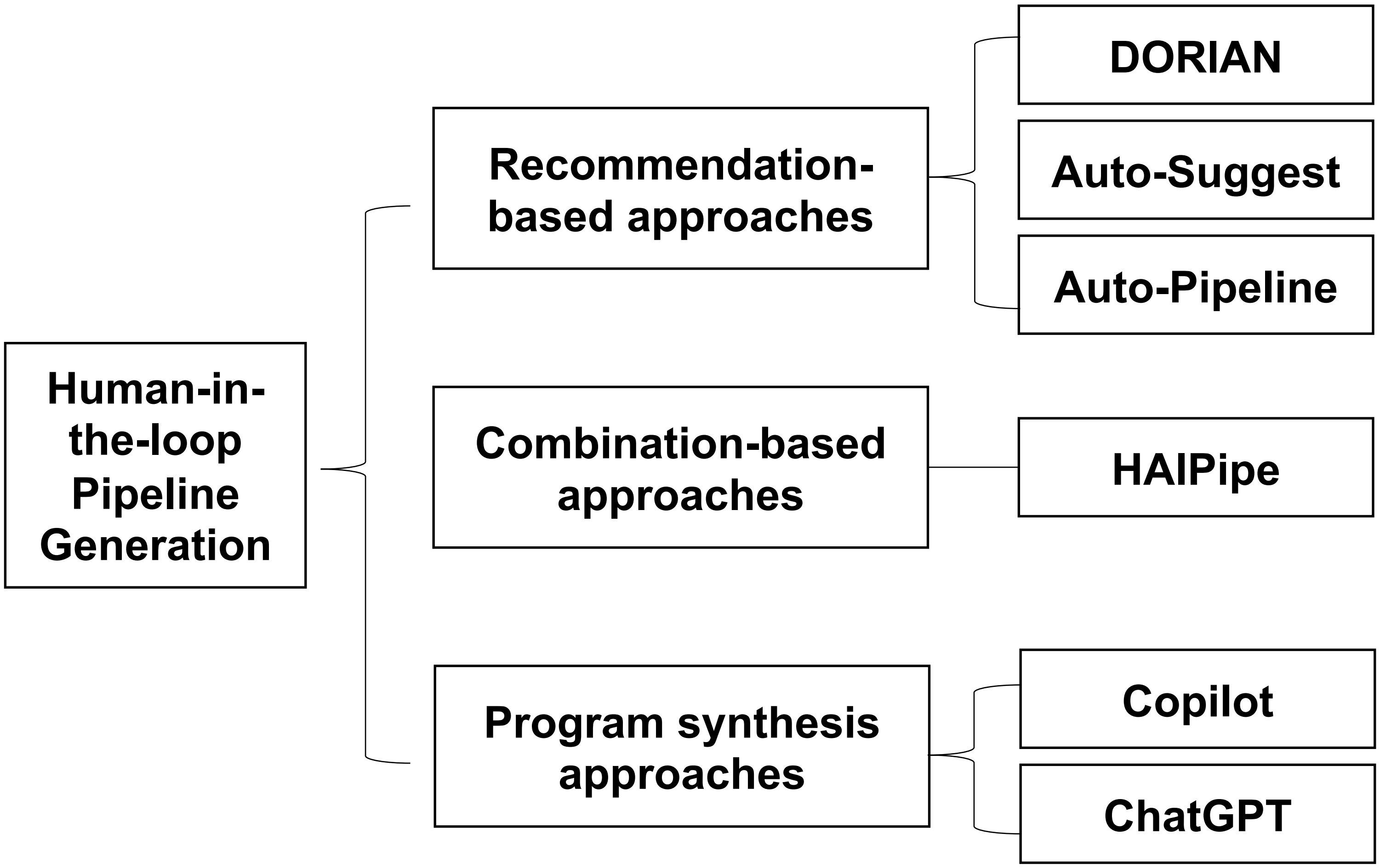
- Relatively low human effort
- Inject the users' feedback and expertise

Outline

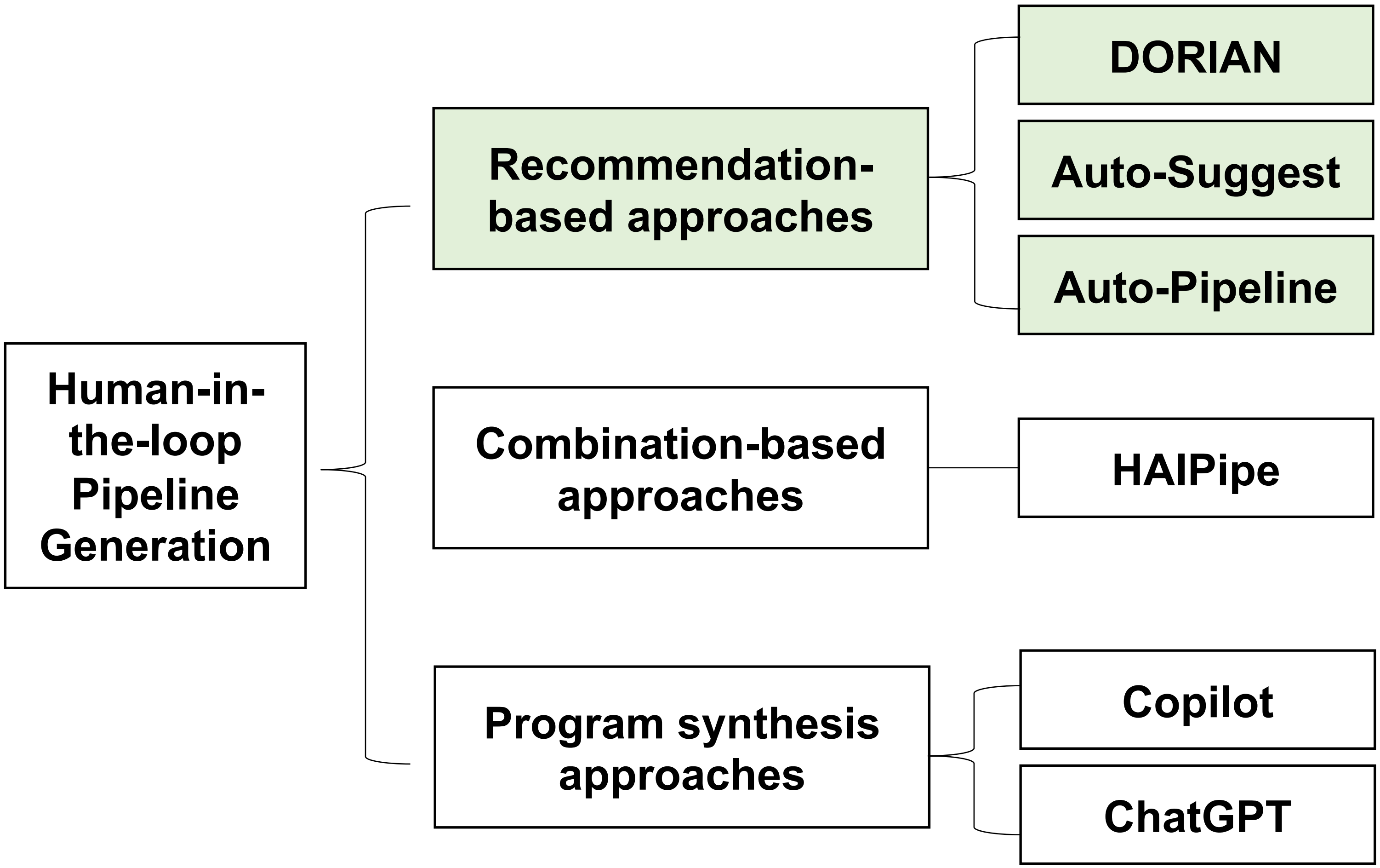
- ☐ Overview
  - Motivation
  - Challenges
  - Manual Pipeline Orchestration
  - Automatic Pipeline Generation
  - Human-in-the-loop Pipeline Generation

- ☐ Open Problems

Human-in-the-loop Pipeline Generation



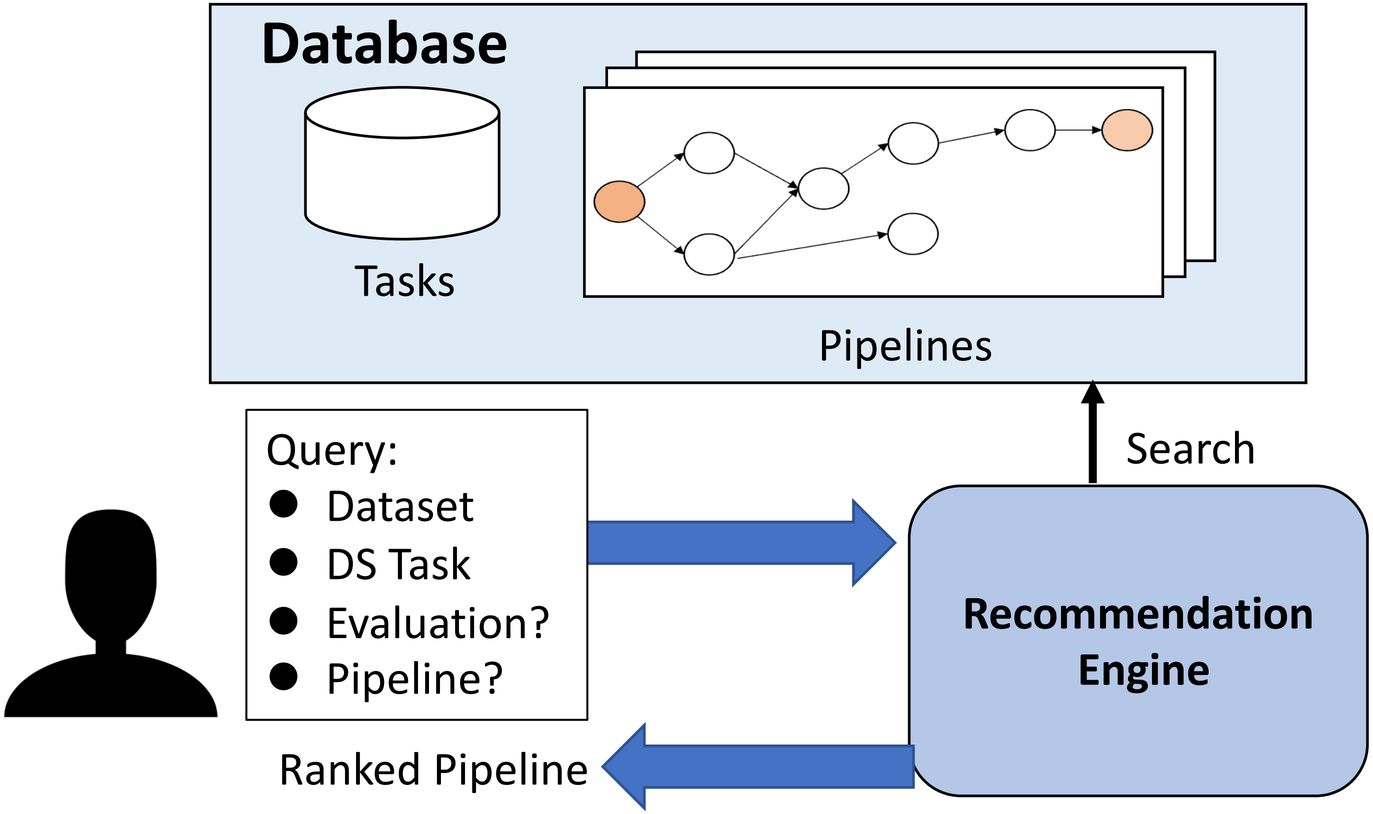
Human-in-the-loop Pipeline Generation



DORIAN

Key Idea

- Offline: a database to store previously pipelines from different teams
- Online: suggest top-k pipelines based on user inputs



[8] Sergey Redyuk et al. DORIAN in action: Assisted Design of Data Science Pipelines. PVLDB 2022

Auto-Suggest

☐ Goal

- Recommend Data Preparation Steps

☐ Key Steps

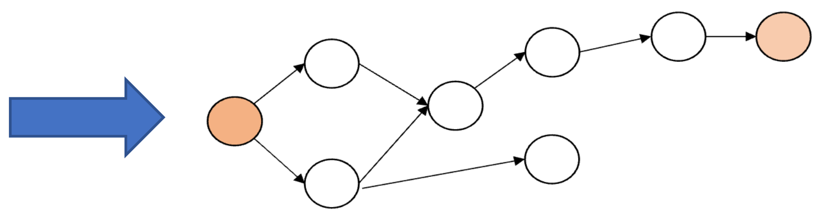
- Data Collection: Python Notebooks from Kaggle/OpenML/Github
- Pipeline Extractor:
  - Python AST Module

```python
X = df.drop("No-show",axis=1)
y = df["No-show"]

X_train1 = pd.get_dummies(X)
y.replace("No", 0,inplace=True)
y.replace("Yes", 1,inplace=True)

scaler = StandardScaler().fit(X_train1)
rescaledX2 = scaler.transform(X_train1)

X_train, X_test, y_train, y_test = train_test_split(
    rescaledX2, y, train_size=0.8, test_size=1-0.8, random_state=0)
```



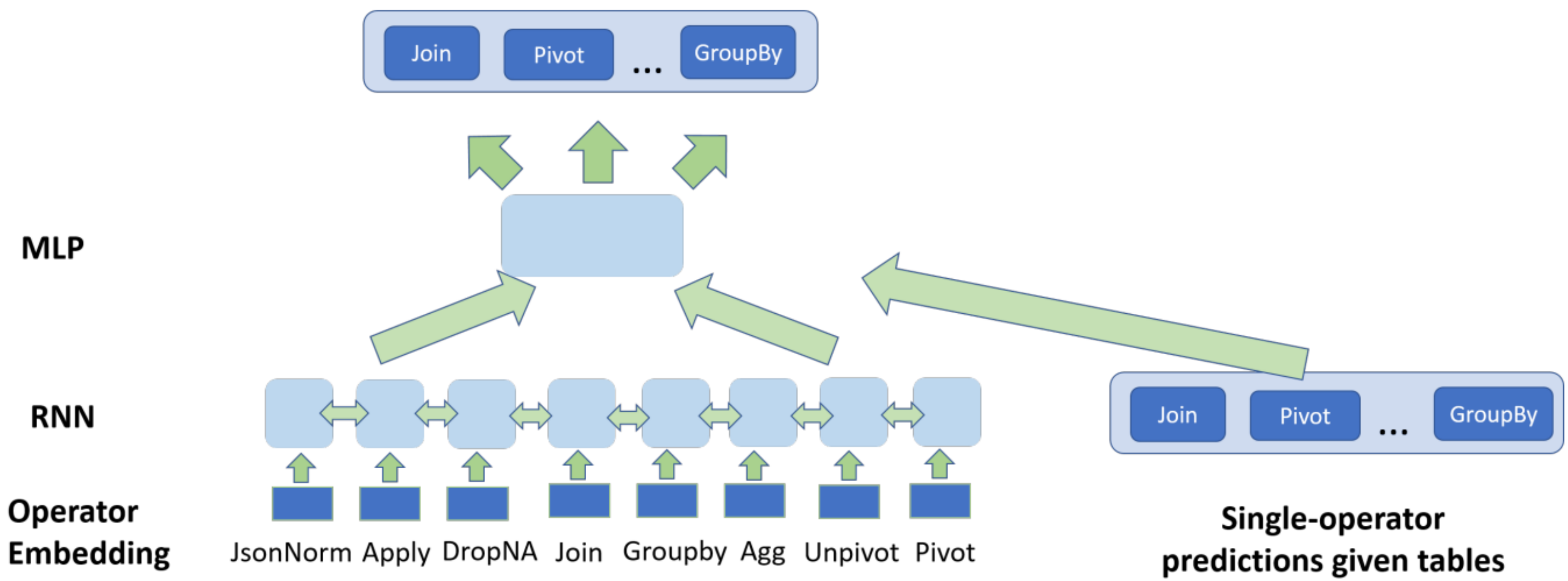
- Pipeline Replay:
  - Handling Missing Packages
  - Handling Missing Data Files

[9] Auto-Suggest: Learning-to-Recommend Data Preparation Steps Using Data Science Notebooks. SIGMOD 2020.

Auto-Suggest

☐ Key Component

- ➢ RNN-based Model

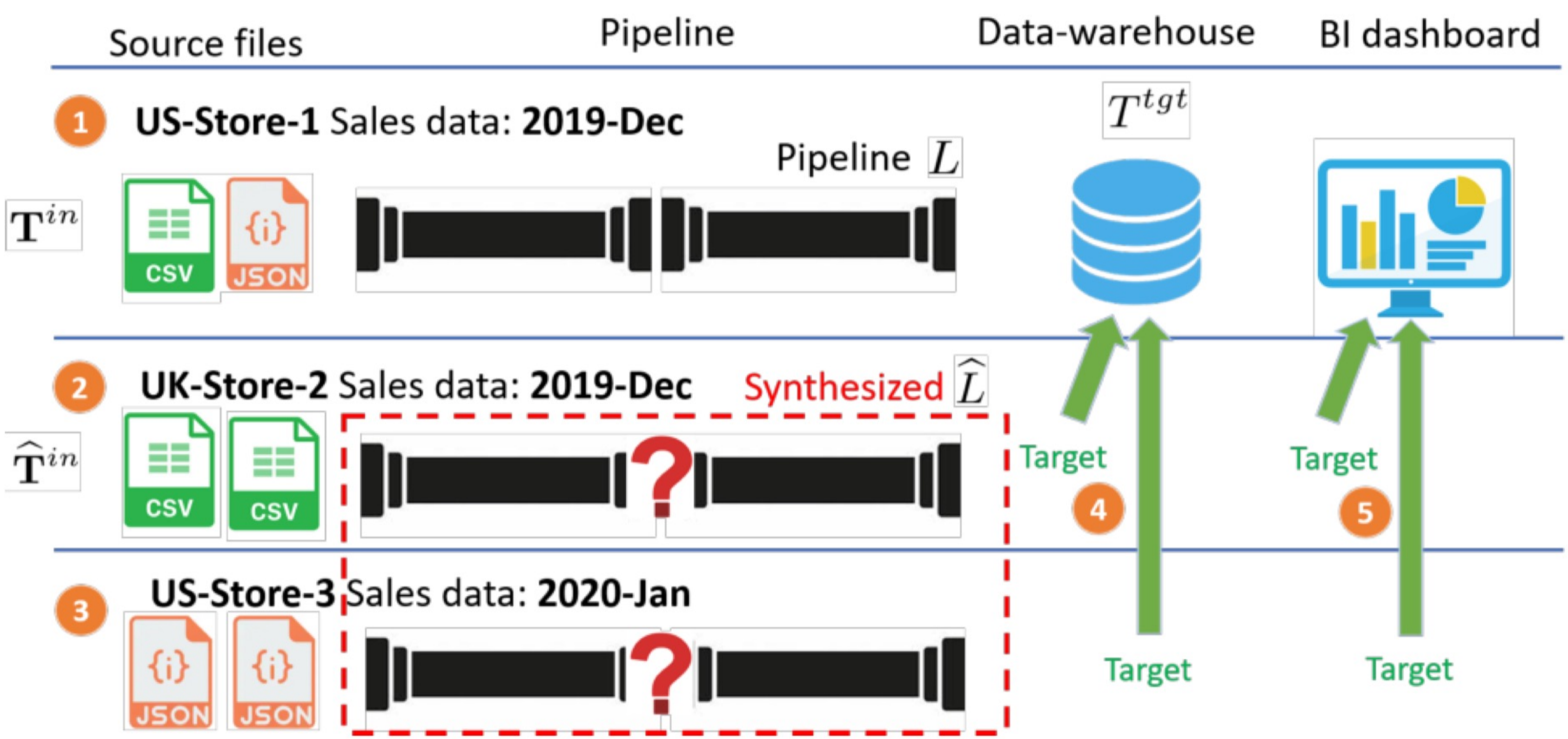


[9] Auto-Suggest: Learning-to-Recommend Data Preparation Steps Using Data Science Notebooks. SIGMOD 2020.

Auto-Pipeline

☐ Key Idea

➢ Automatically Synthesize pipelines from **by-targe (Synthesize by Example)**



[10] Auto-Pipeline: Synthesizing Complex Data Pipelines By-Target Using Reinforcement Learning and Search. VLDB 2021.

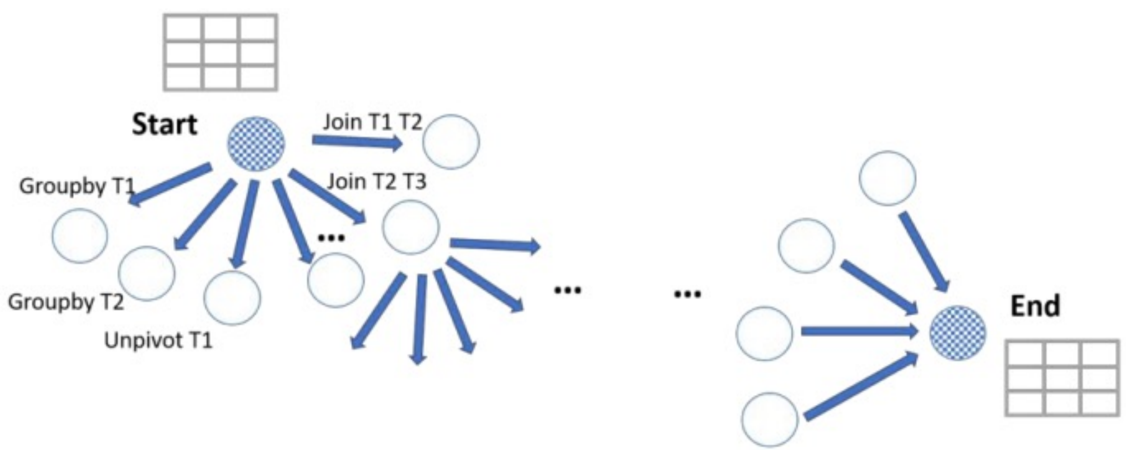
Auto-Pipeline

◻ Framework

- ➢ Input:
  - ➢ a few input dataset to be processed.
  - ➢ an example “target” output.
- ➢ Output: a synthesized pipeline to generate results like the “target”.

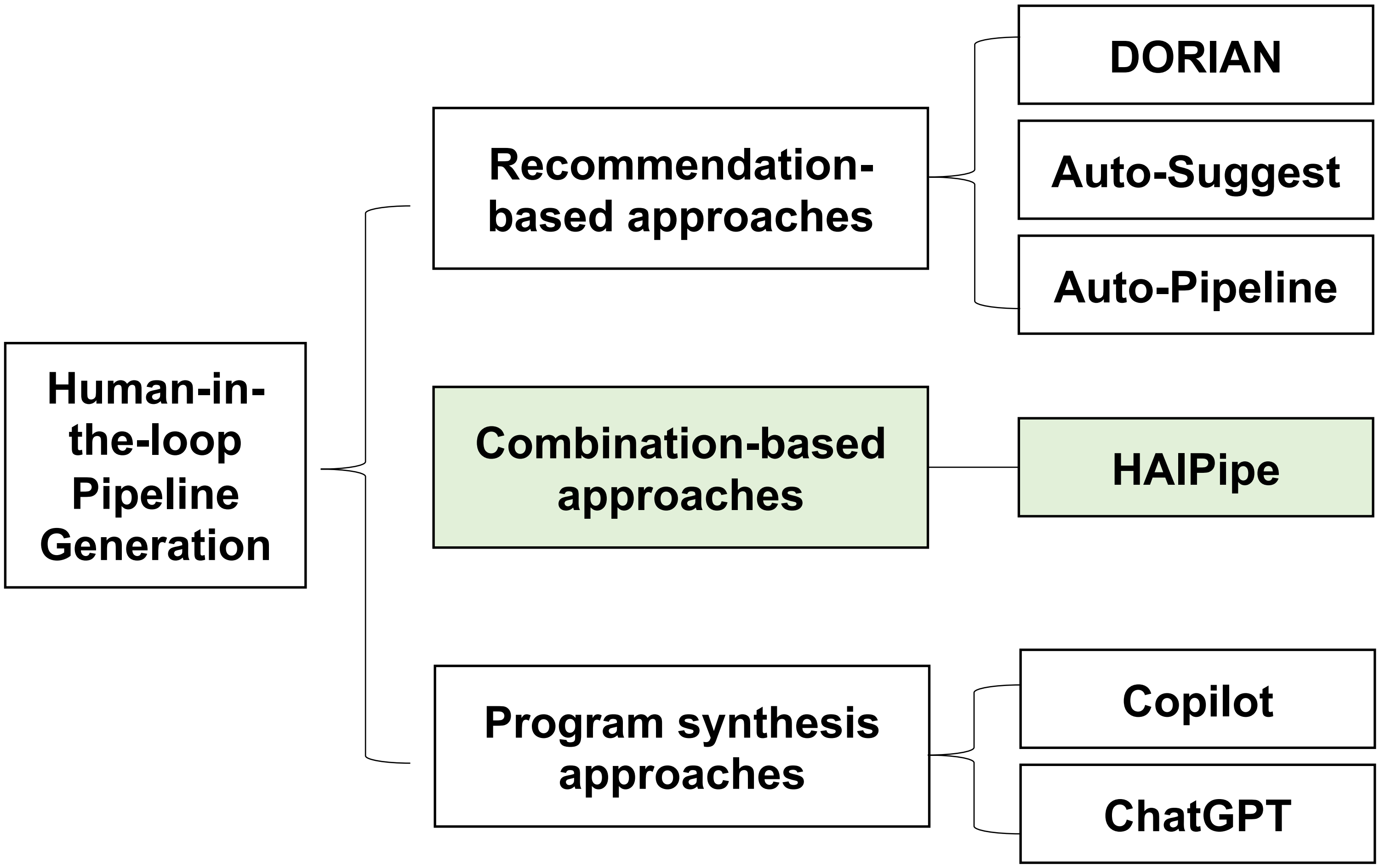
◻ Two Methods:

- ➢ Diversity-based Search among the search space
- ➢ Learn-to-Synthesize by Reinforcement Learning



[10] Auto-Pipeline: Synthesizing Complex Data Pipelines By-Target Using Reinforcement Learning and Search. VLDB 2021.

Human-in-the-loop Pipeline Generation



Combination-based Approaches

	Pros	Cons
Manual Pipeline	**Domain knowledge**	Experience- and heuristic-based
Automatic Pipeline	**Automatic Searching and Generation**	Lack domain knowledge

HAIPipe: can we combine manual pipeline (HI-pipeline) and automatic pipeline (AI-pipeline) to get a new pipeline (HAI-pipeline) that is better than both two pipeline?

[11] HAIPipe: Combining Human-generated and Machine-generated Pipelines for Data Preparation. SIGMOD 2023.

HAIPipe

☐ An Running Example of HAIPipe

```python
import pandas as pd
from sklearn.preprocessing import StandardScaler

data = pd.read_csv("adult.csv")
```

HI-pipeline h_1

```python
data = data[data['workclass'] != '?']
data = data[data['occupation'] != '?']

X = data.drop(["income"], axis=1)
```

AI-pipeline a_1

```python
from sklearn.preprocessing import OneHotEncoder
def one_hot_encoder(data): ⋯
X = one_hot_encoder(X)
```

AI-pipeline a_2

```python
from sklearn.preprocessing import PolynomialFeatures
def polynomial_features(data): ⋯
X = polynomial_features(X)
```

HI-pipeline h_2

```python
y = data["income"]

scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)
```

AI-pipeline a_3

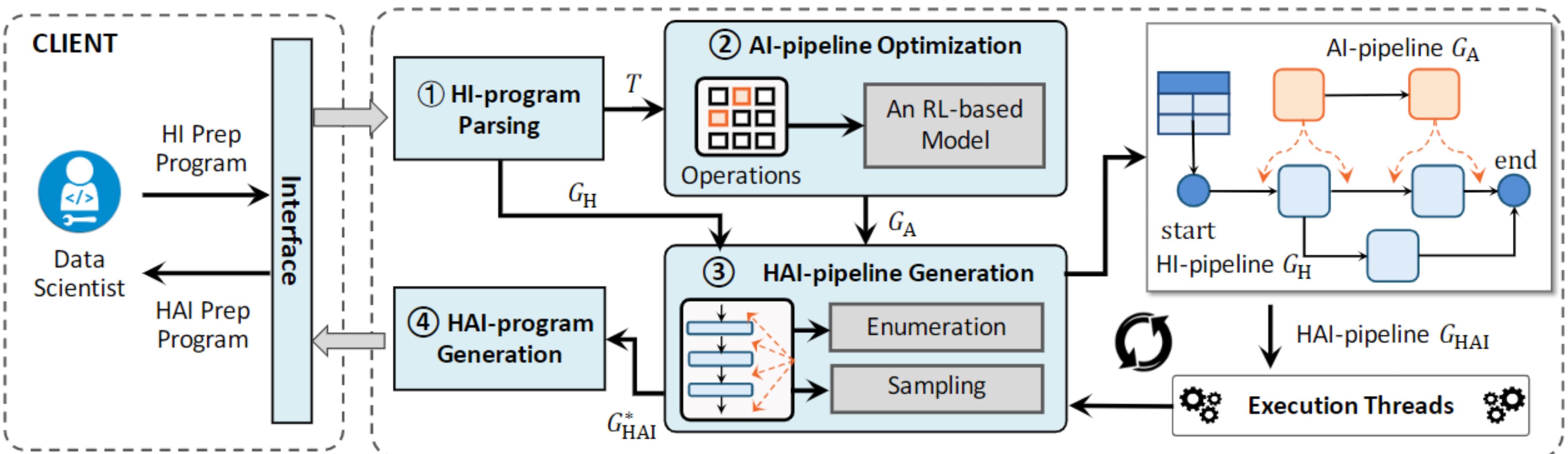
```python
from sklearn.feature_selection import VarianceThreshold
def variance_threshold(data): ⋯
X = variance_threshold(X)
```


[11] HAIPipe: Combining Human-generated and Machine-generated Pipelines for Data Preparation. SIGMOD 2023.

HAIPipe

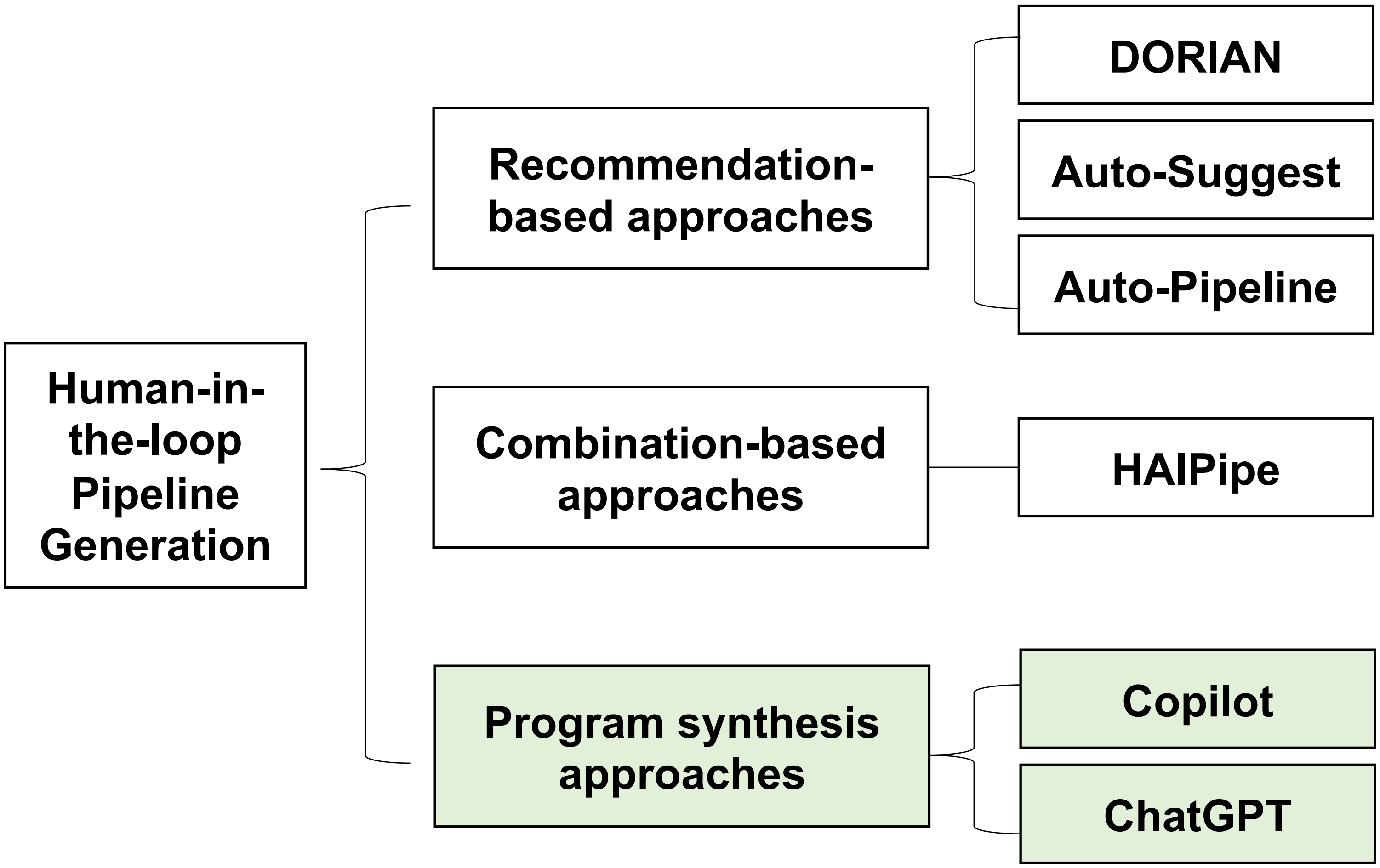
☐ HAIPipe Framework

- ➢ Step1 - HI-program Parsing
- ➢ Step2 - AI-pipeline Optimization
- ➢ Step3 - HAI-pipeline Generation
- ➢ Step4 - HAI-program Generation



[11] HAIPipe: Combining Human-generated and Machine-generated Pipelines for Data Preparation. SIGMOD 2023.

Human-in-the-loop Pipeline Generation



Program Synthesis Approaches

□ Copilot

sentiments.ts | write_sql.go | parse_expenses.py | addresses.rb

```python
import datetime

def parse_expenses(expenses_string):
    """Parse the list of expenses and return the list of triples (date, value, currency).
    Ignore lines starting with #.
    Parse the date using datetime.
    Example expenses_string:
        2016-01-02 -34.01 USD
        2016-01-03 2.59 DKK
        2016-01-03 -2.72 EUR
    """
    expenses = []
    for line in expenses_string.splitlines():
        if line.startswith("#"):
            continue
        date, value, currency = line.split(" ")
        expenses.append((datetime.datetime.strptime(date, "%Y-%m-%d"),
                        float(value),
                        currency))
    return expenses
```

Copilot

Program Synthesis Approaches

☐ ChatGPT

Writing programs through dialogue.

SI please write an ML program for "Diabetes" dataset.

Sure, here's an example machine learning program for the "Diabetes" dataset using Python and scikit-learn library:

python Copy code

```python
# import required libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# load the dataset
data = pd.read_csv("diabetes.csv")

# split the data into training and testing sets
X = data.drop(columns=["Outcome"])
y = data["Outcome"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

# scale the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# train the logistic regression model
clf = LogisticRegression()
clf.fit(X_train, y_train)

# predict on the testing set
y_pred = clf.predict(X_test)

# calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

Note: This program assumes that the "diabetes.csv" file is present in the same directory as the Python script. If the file is located elsewhere, you'll need to modify the file path accordingly.

Program Synthesis Approaches

- **Program synthesis approaches for Data Science**
  - Lack dataset information and domain knowledge.

```
# deal outliers
X = X.clip(lower=X.quantile(0.01), upper=X.quantile(0.99), axis=1)
```

(a) Outlier removal code suggested by **Copilot++**.

```
# deal outliers
columns_nozero_values = \
['Glucose','BloodPressure', 'SkinThickness','Insulin','BMI']
for n in columns_nozero_values:
    data[n] = data[n].replace(0,np.NaN)
    mean = int(data[n].mean())
    data[n] = data[n].replace(np.NaN,mean)
```

(b) Outlier removal code in HI-program written by users.

Outline

- □ Overview
  - Motivation
  - Challenges
  - Manual Pipeline Orchestration
  - Automatic Pipeline Generation
  - Human-in-the-loop Pipeline Generation

- □ **Open Problems**

Open Problems

□ Search Space Refinement

➢ How to utilize human guidance to constrict the search space of possible pipelines and define operations that are specific to particular tasks.

□ Domain Knowledge Injection

➢ How to inject domain knowledge to automatic pipeline generation algorithms.

□ Smooth Integration with AutoML

➢ How to smoothly integrate pipeline generation with other AutoML tasks, such as hyperparameter tuning and model selection.