Fast AutoAugment

After AutoAugment

Algorithm

Algorithm 1: Fast AutoAugment

Input : $(\theta, D_{\text{train}}, K, T, B, N)$

Split D_{train} into K-fold data $D_{\text{train}}^{(k)} = \{(D_{\mathcal{M}}^{(k)}, D_{\mathcal{A}}^{(k)})\}$ `// stratified shuffling`

for $k \in \{1, \ldots, K\}$ **do**

  $\mathcal{T}_*^{(k)} \leftarrow \emptyset$, $(D_{\mathcal{M}}, D_{\mathcal{A}}) \leftarrow (D_{\mathcal{M}}^{(k)}, D_{\mathcal{A}}^{(k)})$ `// initialize`

  Train θ on $D_{\mathcal{M}}$

  for $t \in \{0, \ldots, T-1\}$ **do**

    $\mathcal{B} \leftarrow \text{BayesOptim}(\mathcal{T}, \mathcal{L}(\theta|\mathcal{T}(D_{\mathcal{A}})), B)$ `// explore-and-exploit`

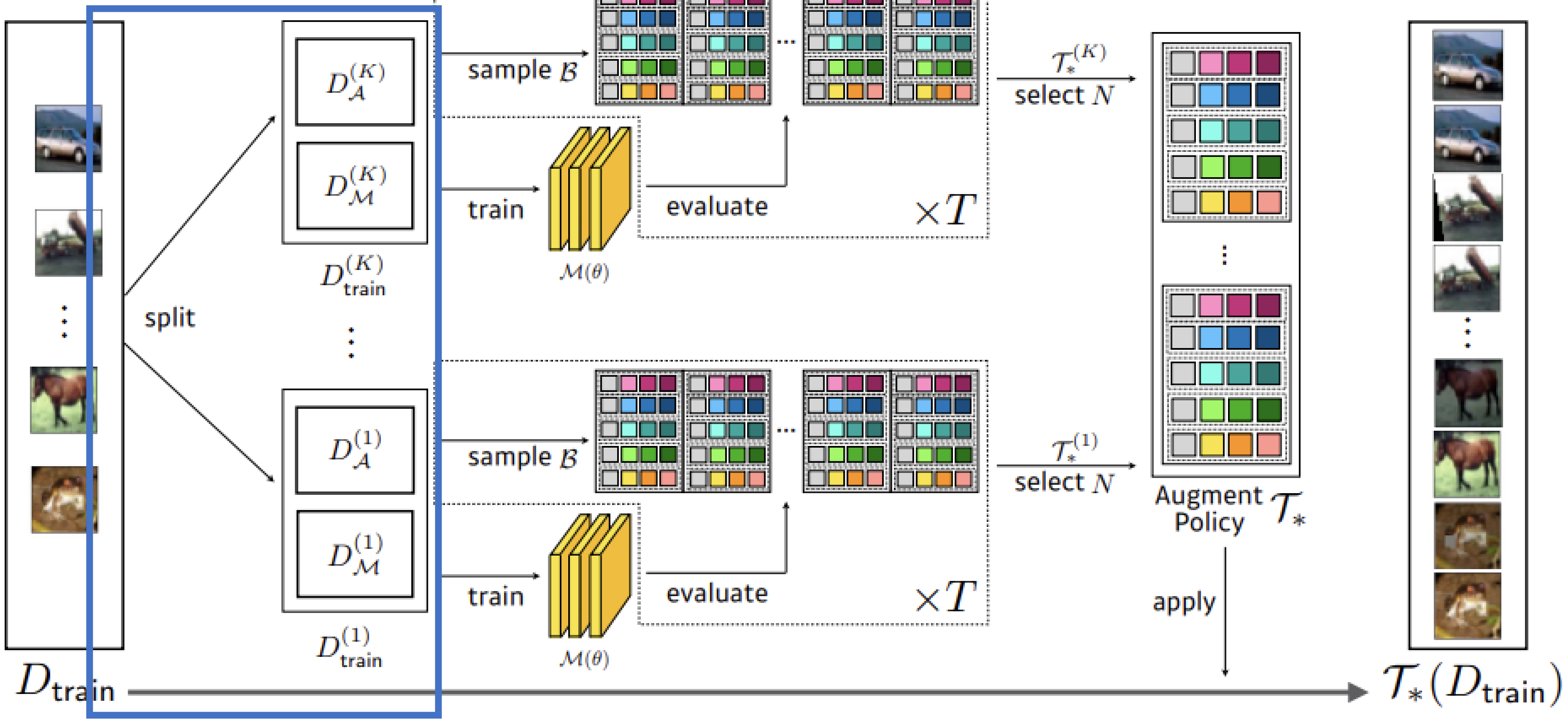
    $\mathcal{T}_t \leftarrow$ Select top-N policies in $\mathcal{B}$

    $\mathcal{T}_*^{(k)} \leftarrow \mathcal{T}_*^{(k)} \cup \mathcal{T}_t$ `// merge augmentation policies`

return $\mathcal{T}_* = \bigcup_k \mathcal{T}_*^{(k)}$

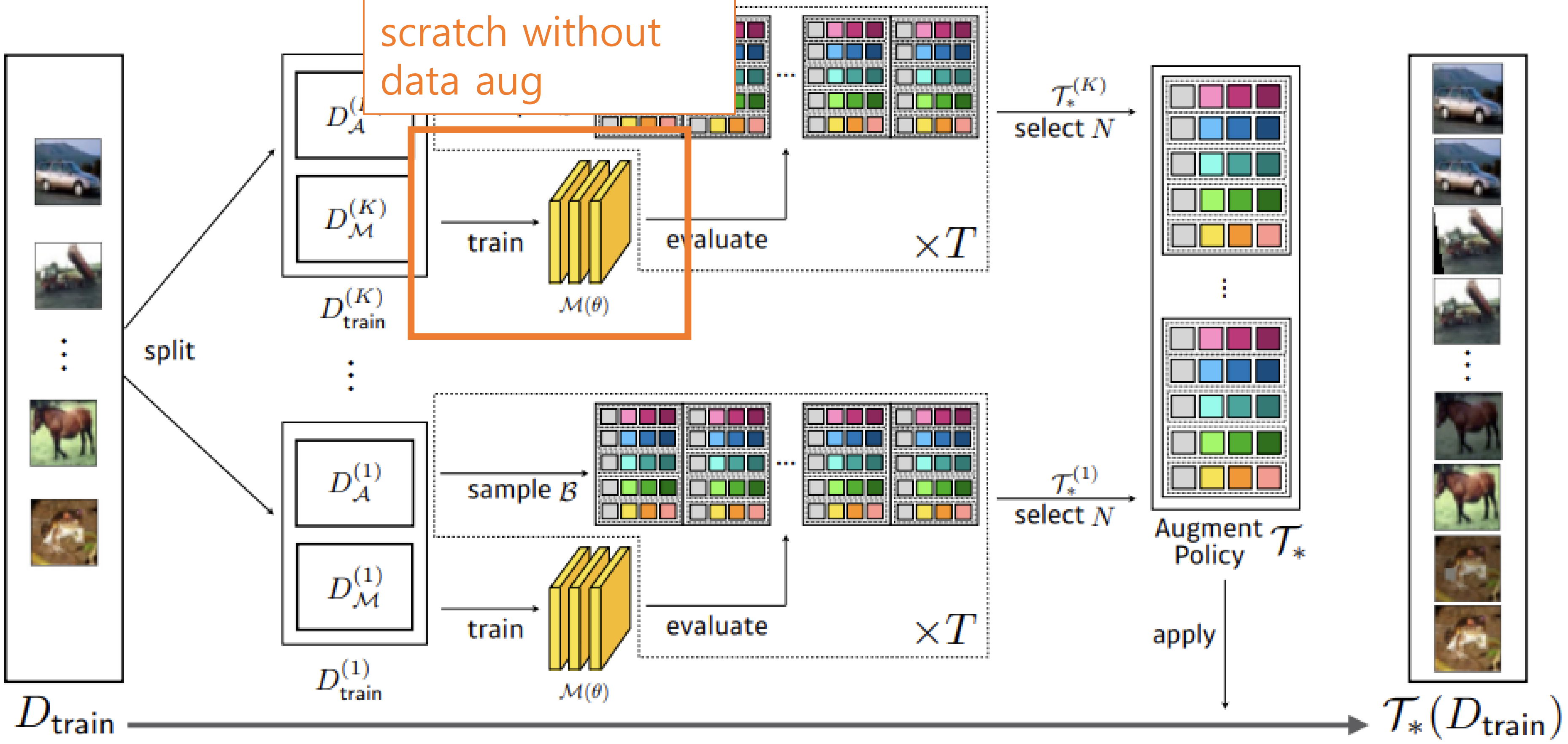
Algorithm Visualization

Step 1: K-fold stratified shuffling



Algorithm Visualization

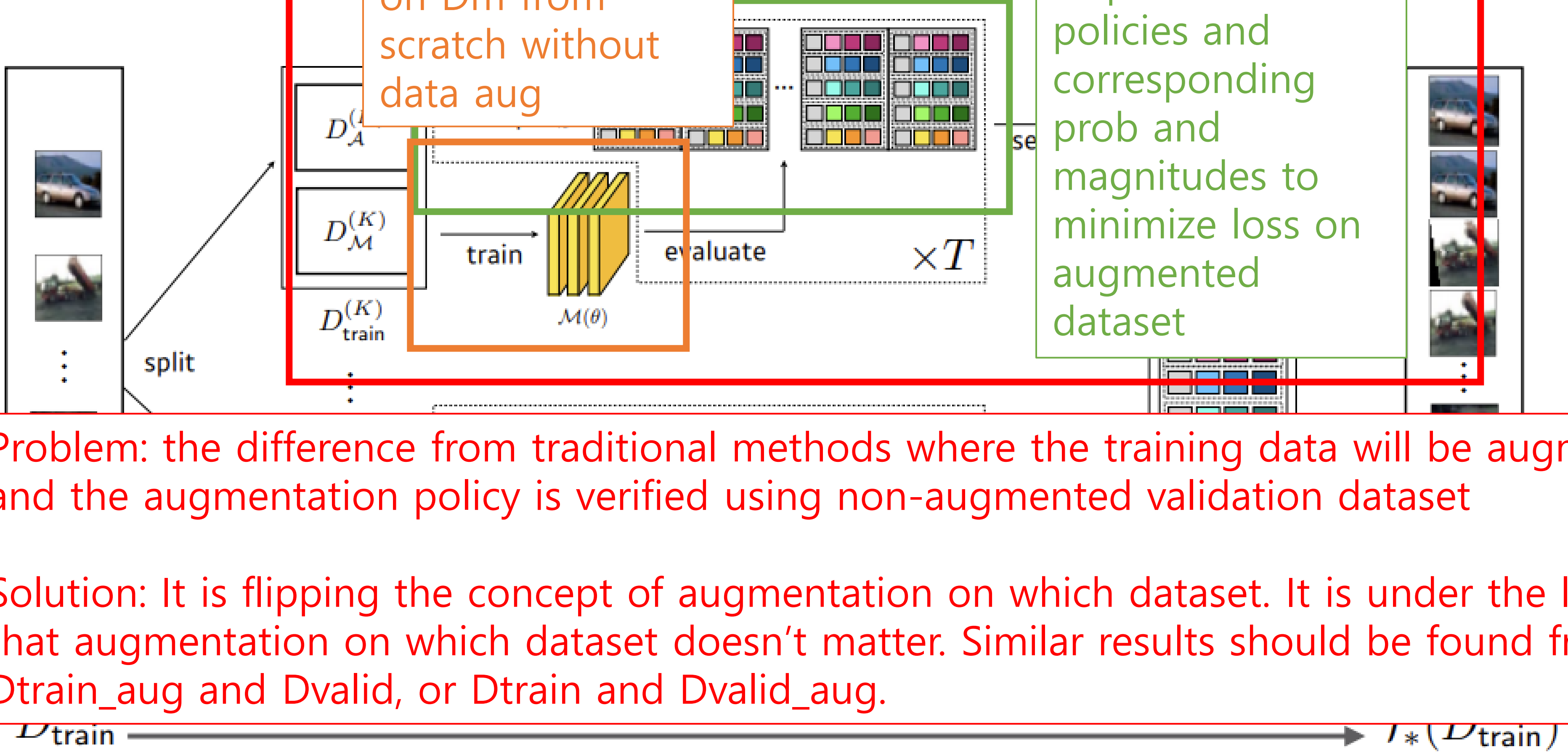
Step 2: train model parameter on Dm from scratch without data aug



Algorithm Visualization

Step 2: train model parameter on Dm from scratch without data aug

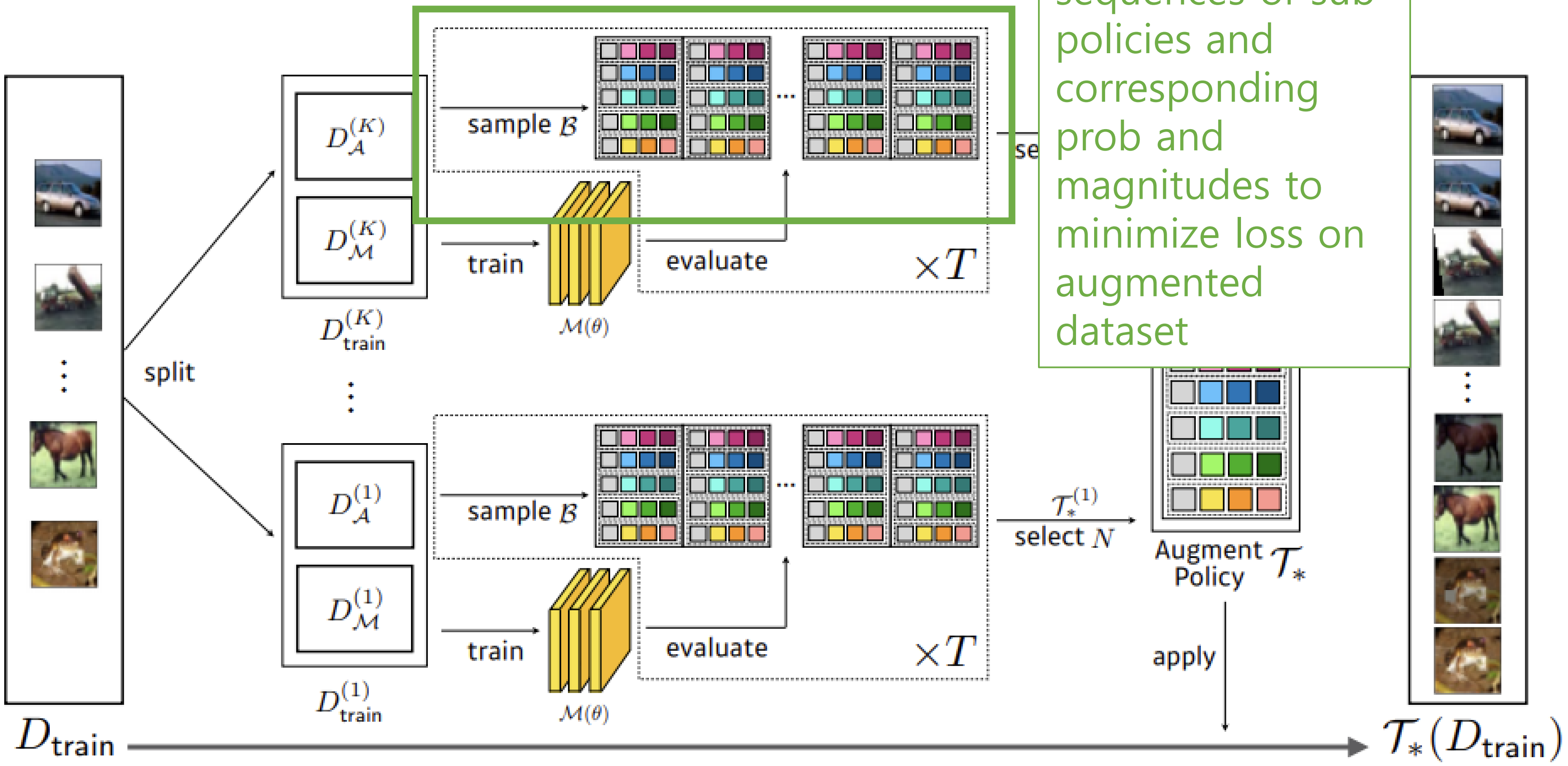
Step 3: Bayesian optimization: repeatedly sample sequences of sub-policies and corresponding prob and magnitudes to minimize loss on augmented dataset



Problem: the difference from traditional methods where the training data will be augmented and the augmentation policy is verified using non-augmented validation dataset

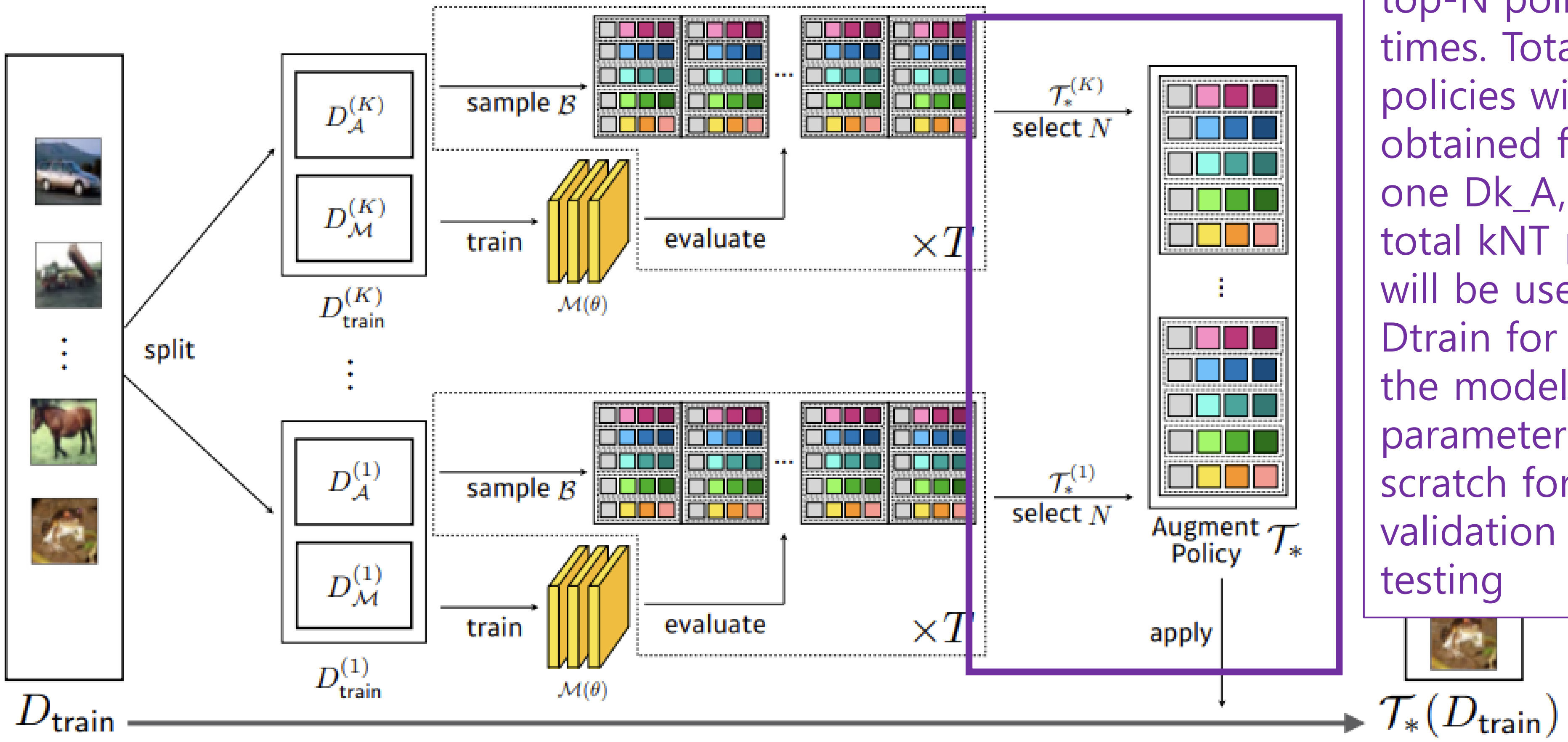
Solution: It is flipping the concept of augmentation on which dataset. It is under the logic that augmentation on which dataset doesn't matter. Similar results should be found from Dtrain_aug and Dvalid, or Dtrain and Dvalid_aug.

Algorithm Visualization



Step 3: Bayesian optimization: repeatedly sample sequences of sub-policies and corresponding prob and magnitudes to minimize loss on augmented dataset

Algorithm Visualization



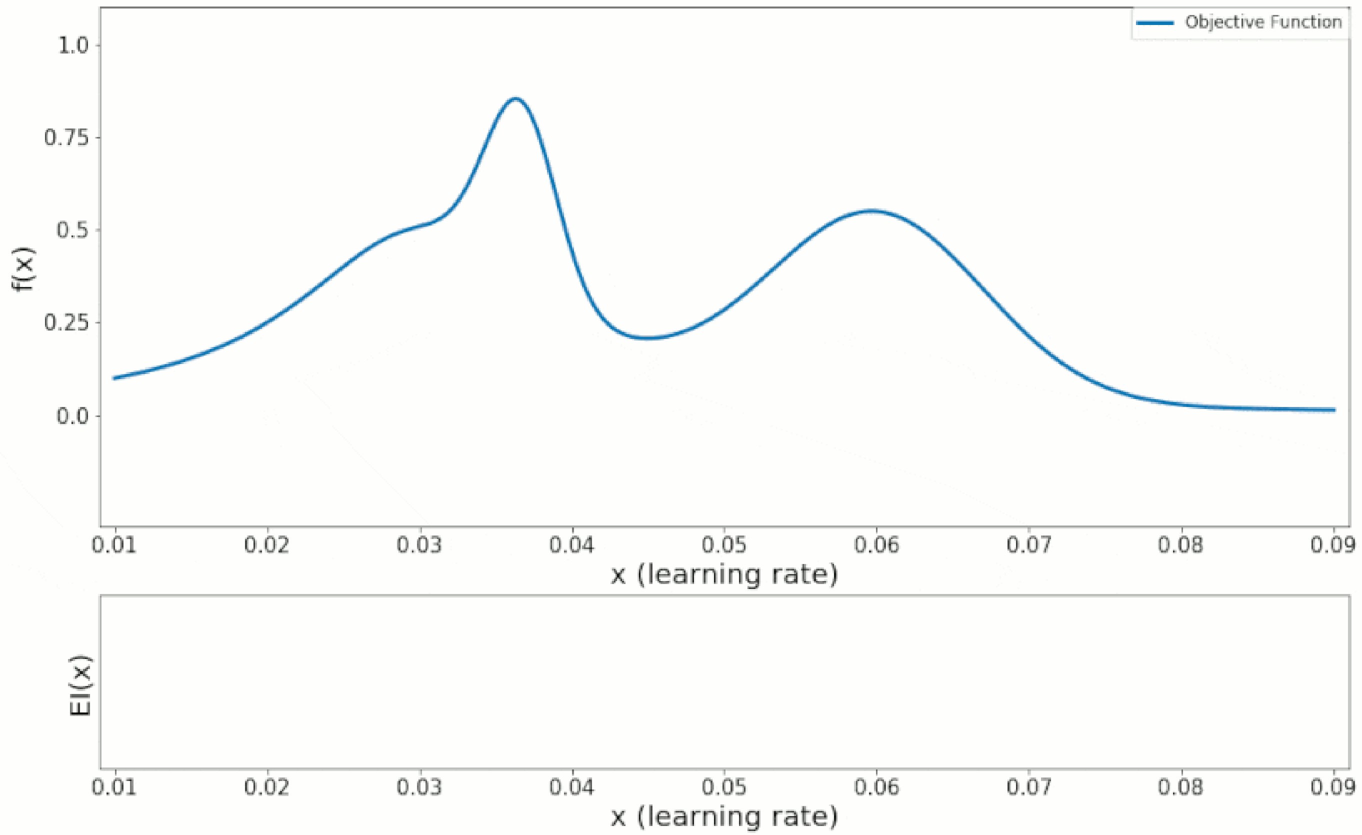
Step 4: Select the top-N policies, T times. Total of NT policies will be obtained from one Dk_A, and in total kNT policies will be used to Dtrain for training the model parameter from scratch for actual validation and testing

Details… Specifications

- 16 operations
- 2 operations per sub-policy
- 5 sub-policy per policy
- 5 fold
- 2 search width (T)
- 10 best policies from search

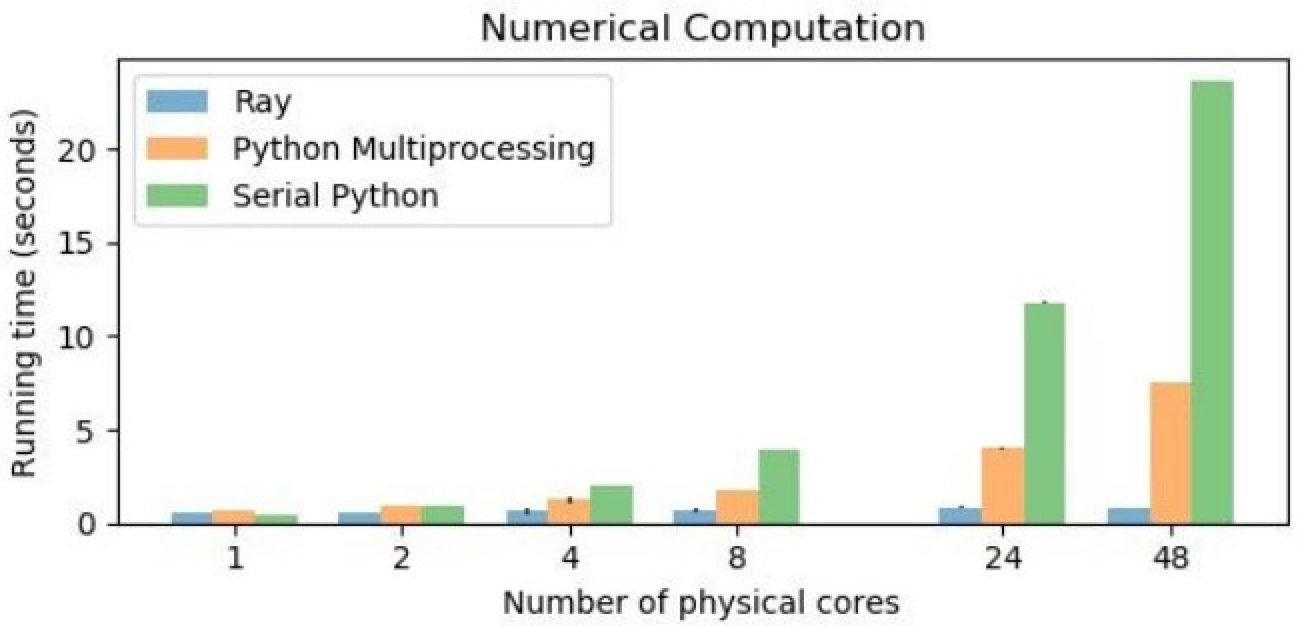
- 100 policies as a result

Details… Bayesian Optimization



https://blog.naver.com/PostView.nhn?blogId=dpfkdlt&logNo=221678800067&from=search&redirect=Log&widgetTypeCall=true&directAccess=false

Details… Ray



Numerical Computation
Ray
Python Multiprocessing
Serial Python
Running time (seconds)
0
5
10
15
20
1
2
4
8
24
48
Number of physical cores

Details... Ray

```python
data = [5, 7, 12, 3, 7, 126, 2, ...]

# Serial Python
def mul(x):
  return x * 10

result = [mul(x) for x in data]

# multiprocessing
def mul(x):
  return x * 10

with multiprocessing.Pool(NUM_CPU) as p:
  result = p.map(mul, data)

# Ray
@ray.remote
def mul(x):
  return x * 10

result = ray.get([mul.remote(x) for x in data])
```

Details… Ray

from ray import tune