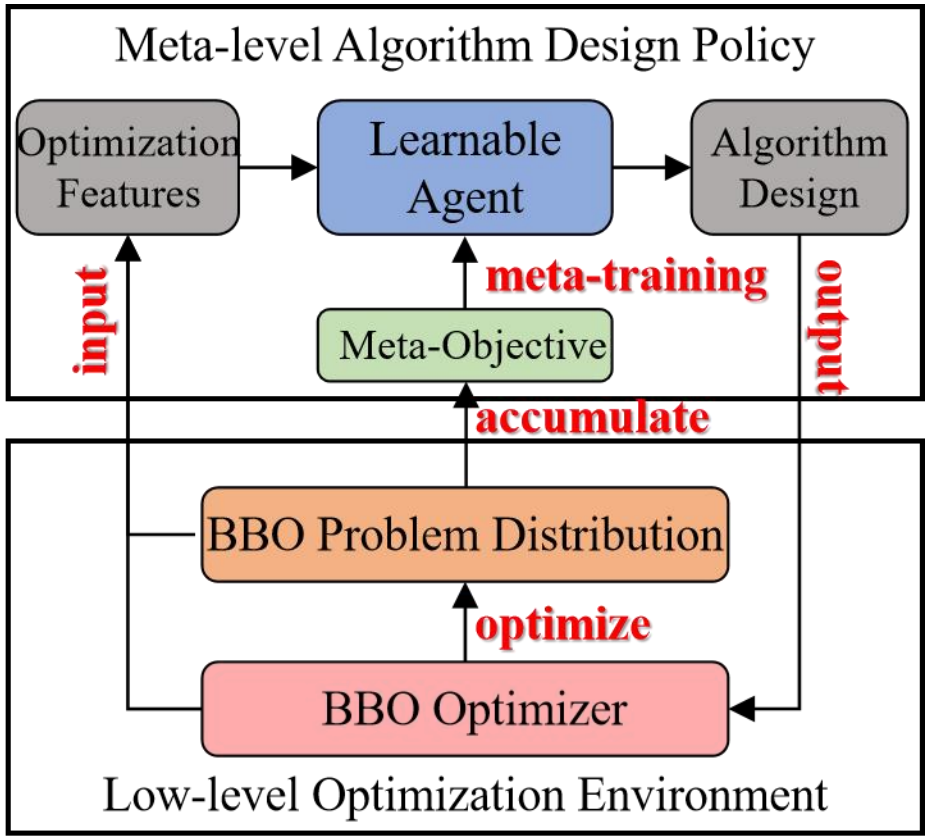
MetaBox-v2: A Unified Benchmark Platform for Meta-Black-Box Optimization

---MA Zeyuan

Background： MetaBBO



Meta-Black-Box Optimization (MetaBBO) and Automated Algorithm Design (AAD) are closely connected. The algorithm scope in MetaBBO can be regarded as a subset of "all algorithms" ---- black-box optimization algorithms.

A learnable agent π_θ is deployed at meta level, it can be a rule-based system, ML algorithm, or neural network.

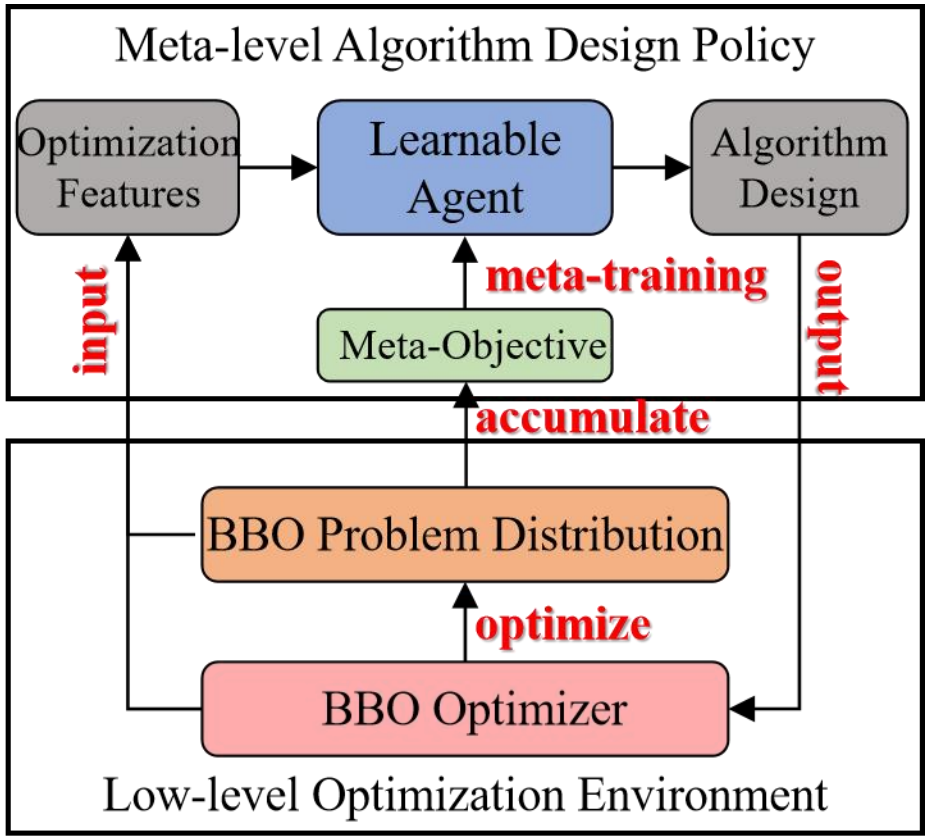
A BBO optimizer A is deployed at lower level, it is used to optimize problem instances in a BBO problem distribution P.

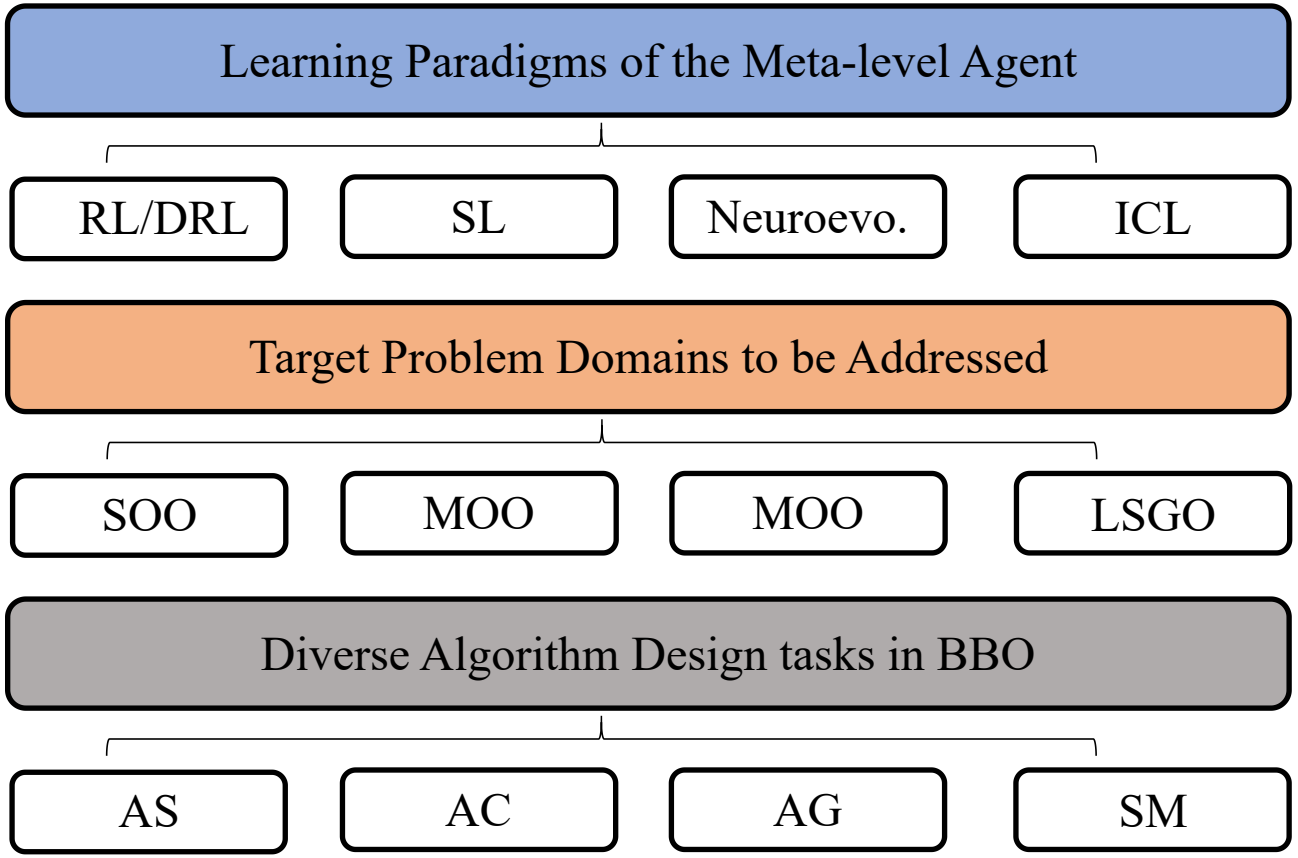
For each optimization step t of the optimization process of each problem instance p, an optimization status s_t could be summarized from current progress. Then a design w_t is tailored by π_θ, which is thereafter assigned to A for current step optimization. A feedback signal $r_{p,t}$ is observed as a measure of design effectiveness.

The meta-objective is then formulated: $J(\theta) = E_{p \epsilon P}[\sum_{t=1}^{T} r_{p,t}]$

[1] Ma, Zeyuan, et al. "Metabox: A benchmark platform for meta-black-box optimization with reinforcement learning." NeurIPS 2023.
[2] Lange, Robert, et al. "Discovering evolution strategies via meta-black-box optimization." ICLR 2023.

Background: MetaBBO





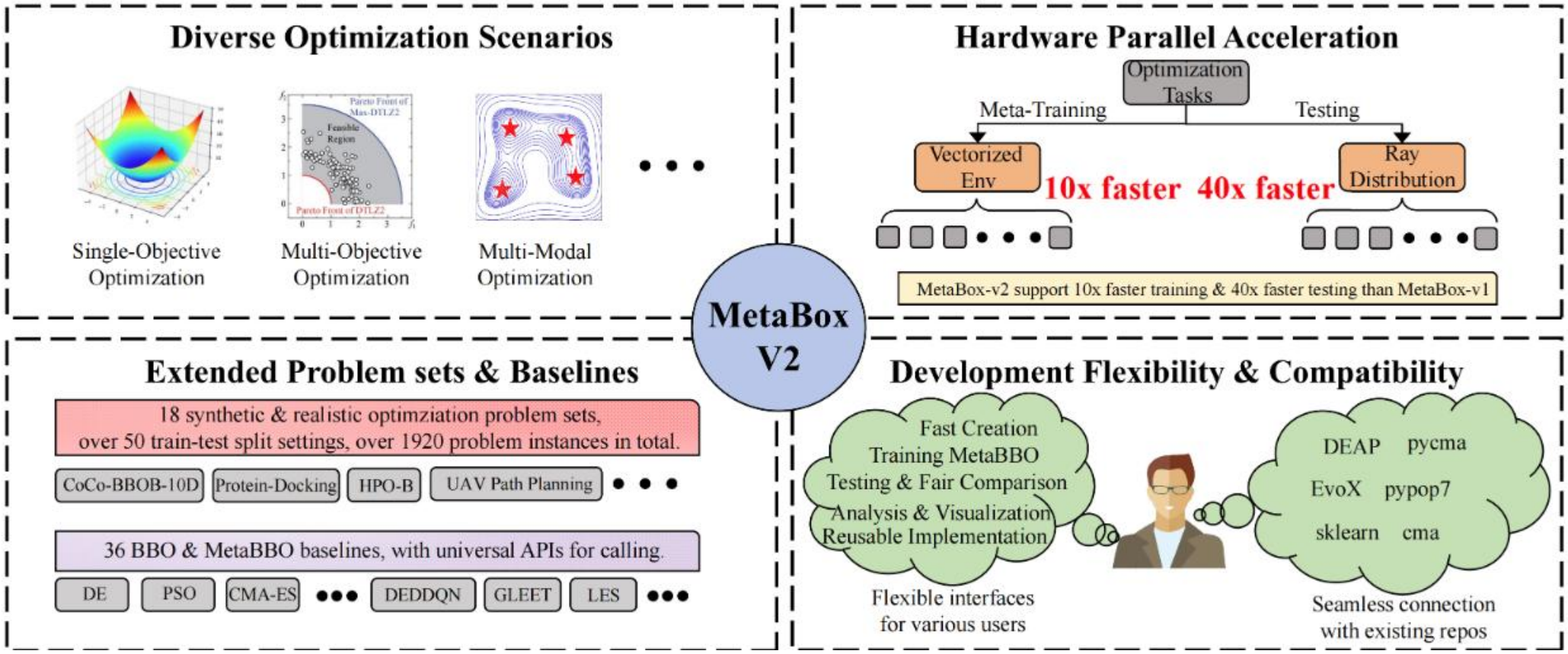
There are more categorization ways, such as the target BBO optimizer, the type of meta-level agent (model-based/free) .

[1] Ma, Zeyuan, et al. "Toward automated algorithm design: A survey and practical guide to meta-black-box-optimization." IEEE TEVC (2025).
[2] Yang, Xu, et al. "Meta-Black-Box optimization for evolutionary algorithms: Review and perspective." SEC (2025).

Implementation: major updates overview

NEURAL INFORMATION PROCESSING SYSTEMS

Four major **user-friendly features** in MetaBox-v2



Implementation: decomposed bi-level architecture

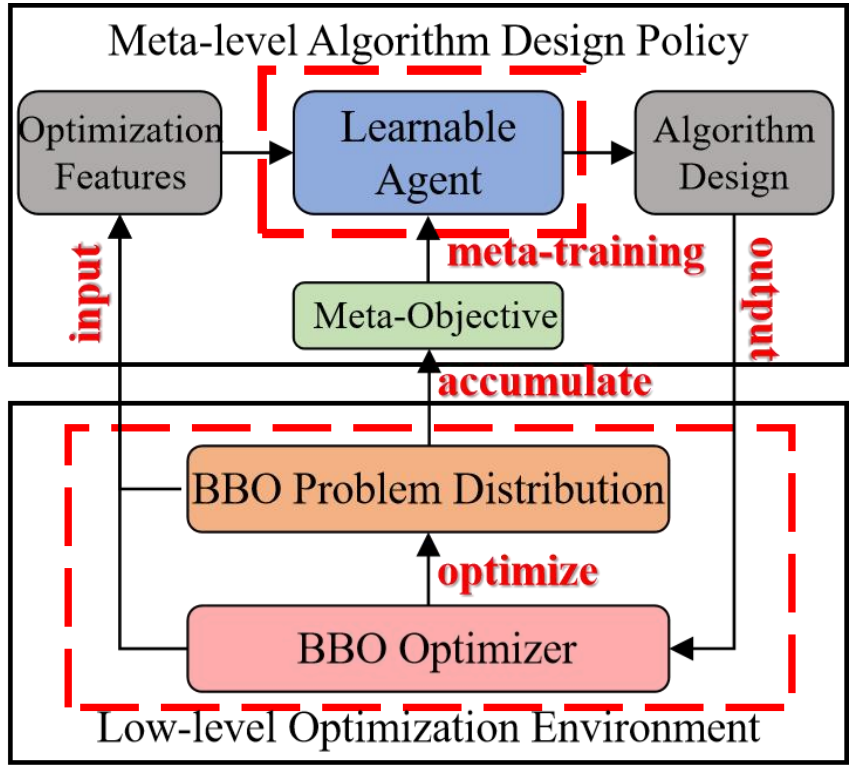
NEURAL INFORMATION PROCESSING SYSTEMS

The bi-level nature of MetaBBO allows us to decompose overall deveopment process with two clearly independent parts

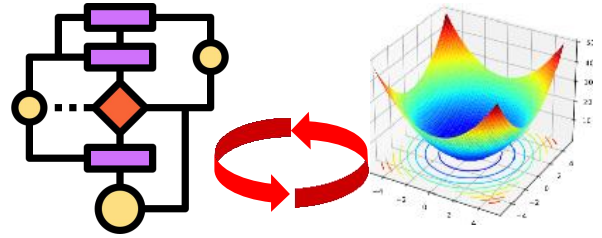
Meta-level Agent

the meta-level agent can be any system that holds certain learning mechanism to refine its behaviour. In MetaBBO, it serves as an independent planner or decision maker to propose desired alg-orithm designfor low-level optimizati-on environment.

Learnable Agent Class



Low-level Optimizer / Optimization Problem



the low-level optimization environment is composed by the algorithm (optimi-zer) and problem instance. In MetaBBO, it is also an independent part, which can be regarded as the dynamics of "world" the meta-level agent could observe. It also provide feedback if needed.

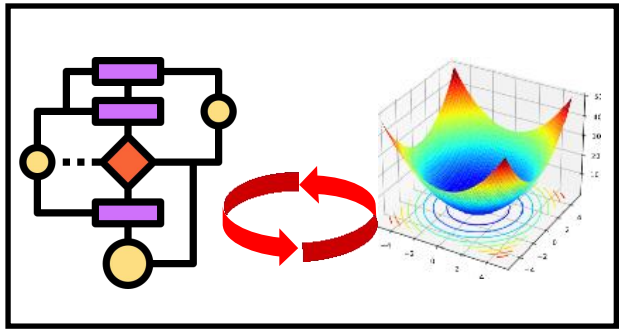
Environment Class

Implementation: universal interface

NEURAL INFORMATION PROCESSING SYSTEMS

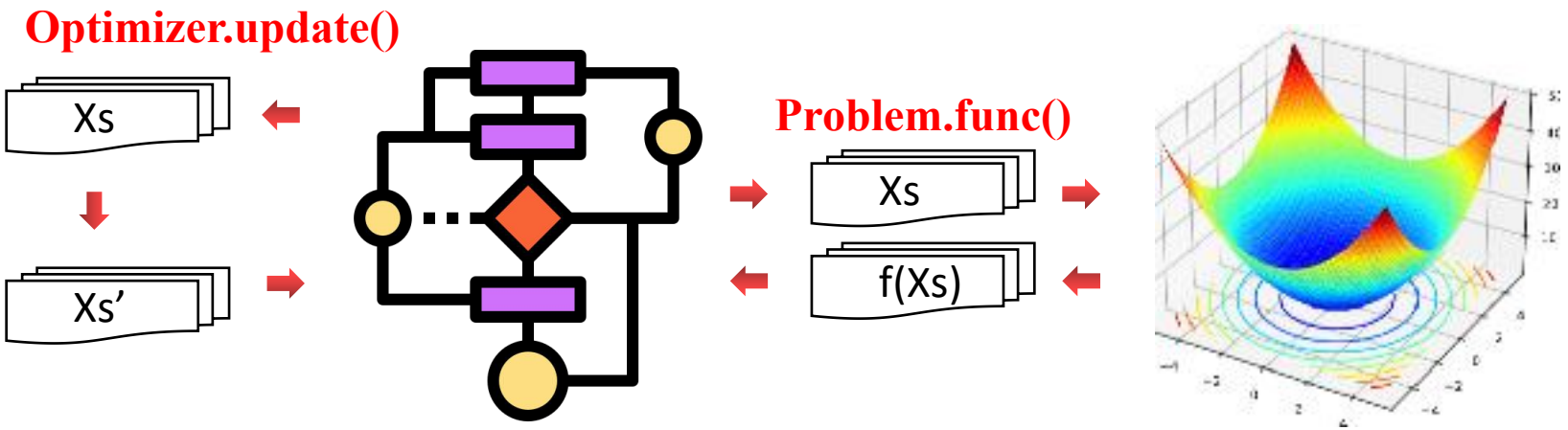
```
Class Learnable_Agent():
    def __init__(self, config):
        # the memory, neural network, learnble parameters
        # architecture etc. are initialized.

    def train_episode(self, env):
        # given the maximum evaluation steps regulated in
        # Environment, this func lets the agent play with the
        # Environment until the optimizer is terminated. It
        # trains the parameters of the agent then.

    def rollout_episode(self, env):
        # same with the train_episode() except that rollout
        # func does not require trace of training signals.
```

```
Class Environment():
    def __init__(self, prob, alg):
        # given the problem instance and the low-level optimizer,
        # the envionment is constructed by composition.

    def reset(self):
        # the optimizer is initialized.

    def step(self, action):
        # once given a specific algorithm design (action) from the
        # meta-level agent, the environment execute it and hence
        # the state such as population in optimzier is updated.
```



```
Class Optimizer():
    def __init__(self, config):
        # initialize hyper-parameters and population.

    def update(self, action,problem):
        # when the Environment receives action, it actually
        # call optimizer's update func to advance the optim-
        # ization by the optimzier's internal logic.
```

```
Class Problem():
    def __init__(self, properties):
        # static properties of a problem instance.

    def func(self, Xs):
        # this func provide evaluation for given solutions.
```



Implementation: diverse data distirbution

NEURAL INFORMATION PROCESSING SYSTEMS

Name	Type	Dimension	maxFEs	Size	Scenario	Description	License
bbob-10D[42]	SOO	10D	2E4	24	synthetic	Single-objective instances in CoCo	BSD-3-Clause
bbob-30D[42]	SOO	30D	5E4	24	synthetic	Single-objective instances in CoCo	BSD-3-Clause
bbob-noisy-10D[42]	SOO	10D	2E4	24	synthetic	*bbob-10D* with gaussian noise	BSD-3-Clause
bbob-noisy-30D[42]	SOO	30D	5E4	24	synthetic	*bbob-30D* with gaussian noise	BSD-3-Clause
hpo-b[63]	SOO	2-16D	2E3	935	realistic	Hyper-parameter optimization	MIT License
uav[65]	SOO	30D	2.5E3	56	realistic	UAV path planning tasks	Attribution 4.0
protein[64]	SOO	12D	2E3	280	realistic	Simplified protein-docking instances	Attribution 4.0
lsgo[60]	LSGO	$\geq$905D	3E6	20	synthetic	Large-scale problem instances	GPL-3.0
ne[51]	LSGO	$\geq$1000D	2.5E3	66	realistic	Neuroevolution for control tasks	GPL-3.0
zdt[56]	MOO	10-30D	5E3	5	synthetic	A group of bi-objective problems	Apache-2.0
uf[58]	MOO	30D	5E3	10	synthetic	Multi-objective problem instances	Apache-2.0
dtlz[57]	MOO	6-29D	5E3	46	synthetic	Scalable multi-objective problems	Apache-2.0
wfg[59]	MOO	12-38D	5E3	117	synthetic	Complex multi-objective problems	Apache-2.0
moo-uav[56]	MOO	30D	2.5E3	56	realistic	Multi-objective form of *uav*	Apache-2.0
mmo[61]	MMO	1-20D	5E4-4E5	20	synthetic	Standard multi-modal problems	Simplified BSD
cec2017mto[62]	MTO	25-50D	2.5E4	9	synthetic	Multi-task problems in CEC2017	-
wcci2020[62]	MTO	50D	6.25E5	10	synthetic	Multi-task problems in WCCI2020	-
wcci2020-aug	MTO	50D	1.25E5	127	synthetic	Flexible combinations of *wcci2020*	-

Compared to MetaBox-v1, we leverage the universal interfaces convenience and have integrated widely used and discussed optimization problem types:

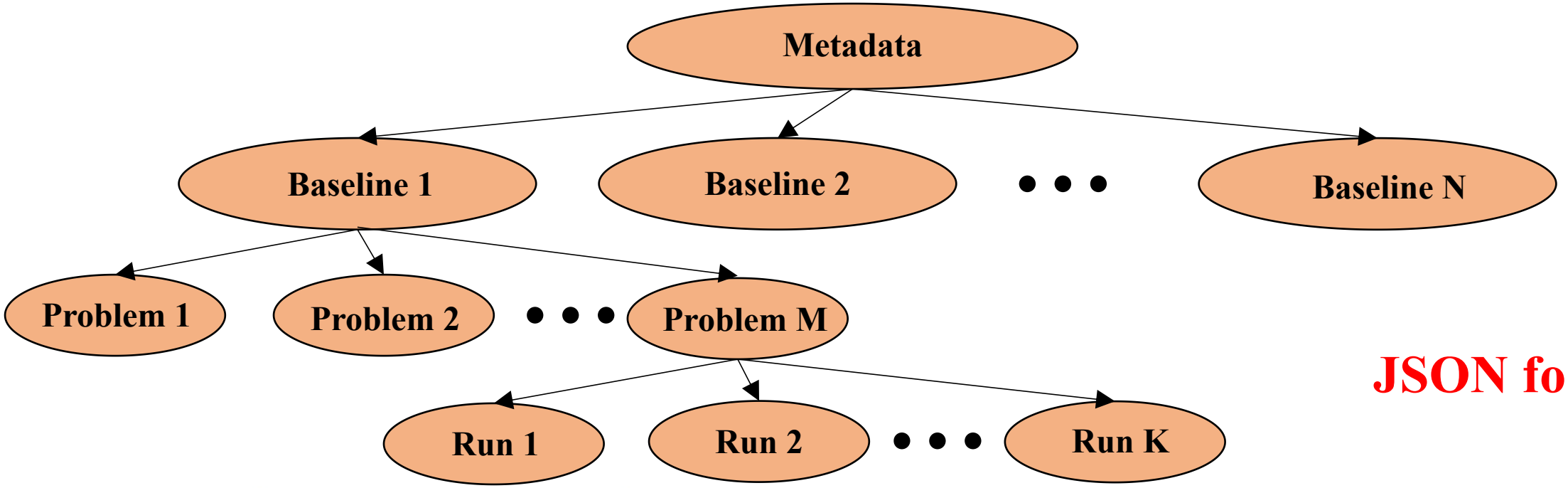
1. Single-Objective Optimization-----**Synthetic**: bbob(-noisy)-10D/30D;
Realistic: hyper-parameter optimization in ML, protein-protein docking and uav path planning.
2. ...

Implementation: flexible meta-data

NEURAL INFORMATION PROCESSING SYSTEMS

We propose a Metadata Object to trace all basic logits obtained when we benchmark the one we developed, sometimes with some other baselines that we want to compare to. The motivation behind roots from the usage feedback from users of MetaBox-v1: "Can we customize our own benchmark evaluation metrics?"

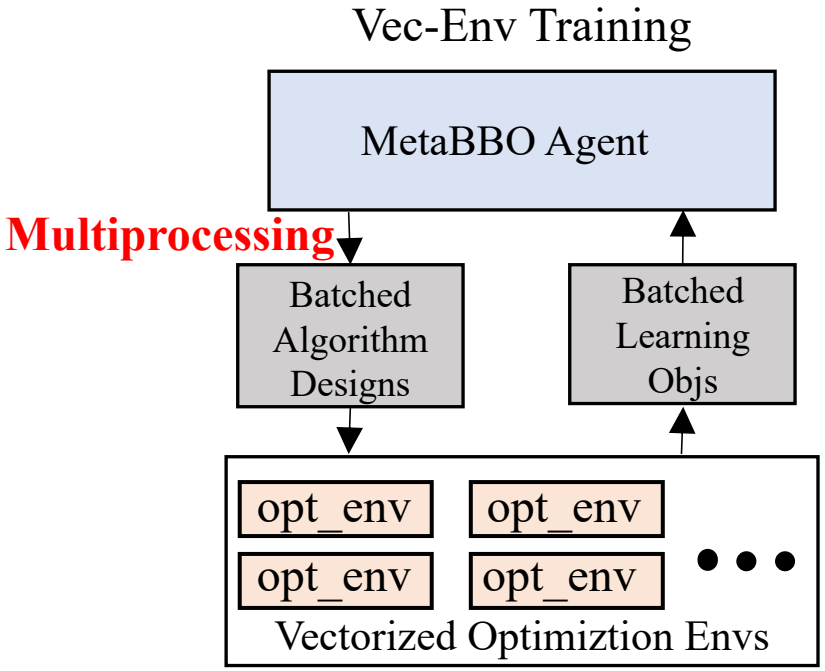
Suppose we have N baselines to benchmark on a problem set with M problems & K independent runs.



JSON format

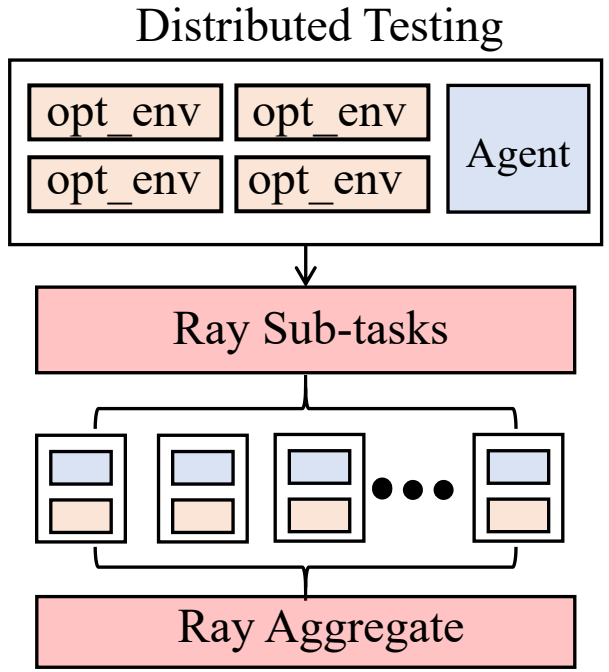
Implementation: parallel train-test

NEURAL INFORMATION PROCESSING SYSTEMS



MetaBBO is a bi-level nested paradigm, which is a critical but understudied bottleneck in current literature.

Serialized environment evaluation in exsiting MetaBBO approaches consumes massive time if the target problem if time-consuming to evluate.



Ray is an very easy-to-use ML distirbution toolbox.

Given an agent and a testsuite, we first copy the agent for each testing run and use Ray to construct the corresponding sub-tasks. Then all sub-tasks are distributed into independent CPU/GPU cores for parallel testing.

4 different modes for users to adapt their hardwares.

If you are a newcomer who wants to learn and try some exsiting MetaBBO approaches, MetaBox-v2 could provide you the fastest way to get started (approximately 10 lines of codes).

```python
from metaevobox import Config, Trainer
# import meta-level agent of MetaBBO you want to meta-train
from metaevobox.baseline.metabbo import GLEET
# import low-level BBO optimizer of MetaBBO you want to meta-train
from metaevobox.environment.optimizer import GLEET_Optimizer
from metaevobox.environment.problem.utils import construct_problem_set

# put user-specific configuration
config = {'train_problem': 'bbob-10D', # specify the problem set you want to train your MetaBBO
          'train_batch_size': 16,
          'train_parallel_mode':'subproc', # choose parallel training mode
          }
config = Config(config)
# construct dataset
config, datasets = construct_problem_set(config)
# initialize your MetaBBO's meta-level agent & low-level optimizer
gleet = GLEET(config)
gleet_opt = GLEET_Optimizer(config)
trainer = Trainer(config, gleet, gleet_opt, datasets)
trainer.train()
```

```python
from metaevobox import Config, Tester, get_baseline
# import meta-level agent of MetaBBO you want to test
from metaevobox.baseline.metabbo import GLEET
# import low-level BBO optimizer of MetaBBO you want to test
from metaevobox.environment.optimizer import GLEET_Optimizer
# import other baselines you want to compare with your MetaBBO
from metaevobox.baseline.bbo import CMAES, SHADE
from metaevobox.environment.problem.utils import construct_problem_set

# specify your configuration
config = {
    'test_problem':'bbob-10D', # specify the problem set you want to benchmark
    'test_batch_size':16,
    'test_difficulty':'difficult', # this is a train-test split mode
    'baselines':{
        # your MetaBBO
        'GLEET':{
            'agent': 'GLEET',
            'optimizer': GLEET_Optimizer,
            'model_load_path': None, # by default is None, we will load a built-in pre-trained checkpoi
        },

        # Other baselines to compare
        'SHADE':{'optimizer': SHADE},
        'CMAES':{'optimizer':CMAES},
    },
}

config = Config(config)
# load test dataset
config, datasets = construct_problem_set(config)
# initialize all baselines to compare (yours + others)
baselines, config = get_baseline(config)
# initialize tester
tester = Tester(config, baselines, datasets)
# test
tester.test()
```

MetaBox-v2 has certain advantages compared to related benchmarks:

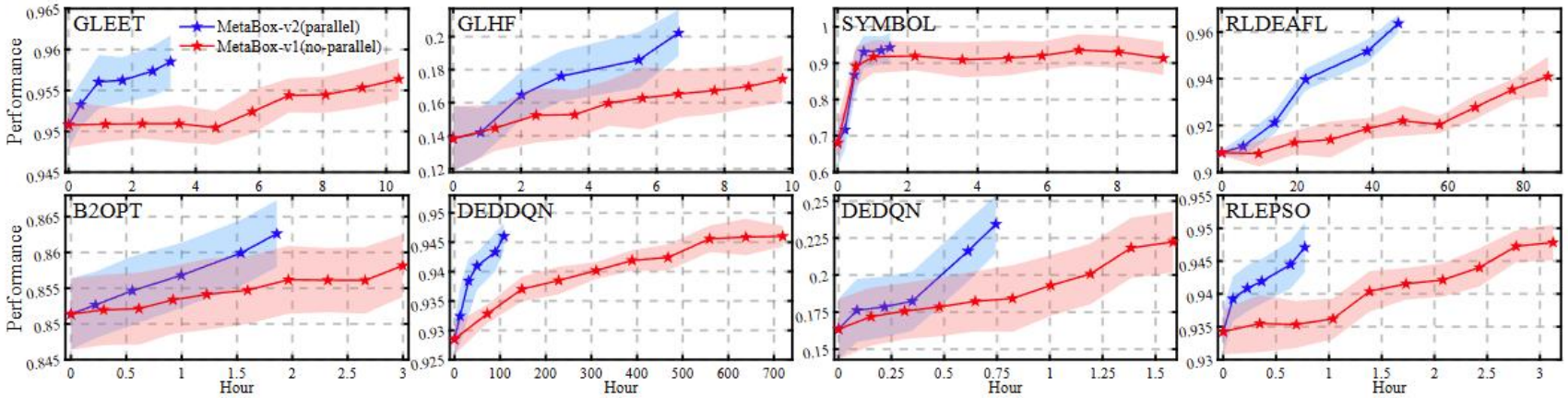
	#Optimization Scopes	*Learning Support*	*Parallel*	*#Problems*	*#MetaBBO Baselines*	*Template*	*Auto*	*Custom*	*Visual*	*Compatibility*
COCO [42]	4	×	×	481+0	×	✓	✓	×	✓	few
CEC [43]	1	×	×	30+0	×	×	×	×	×	none
IOHprofiler [44]	3	×	×	55+0	×	✓	×	✓	✓	few
Bayesmark [45]	1	×	×	0+228	×	✓	✓	×	×	few
Zigzag [46]	1	×	×	4+0	×	×	×	✓	×	none
Engineering [47]	1	×	×	+57	×	×	×	×	×	none
MA-BBOB [48]	1	×	×	1000+0	×	×	×	×	×	none
BBOPlace [49]	1	×	×	0+14	×	×	✓	×	×	none
PyPop7 [50]	2	×	×	92+11	×	×	×	×	×	few
EvoX [51]	2	×	✓	44+50	×	✓	×	✓	✓	rich
MetaBox [8]	1	RL	×	54+280	8	✓	✓	✓	✓	few
MetaBox-v2	5	RL,SL, NE,ICL	✓	541+1393	23	✓	✓	✓	✓	rich

However, benchmarking development is not a “Zero-Sum Game”!

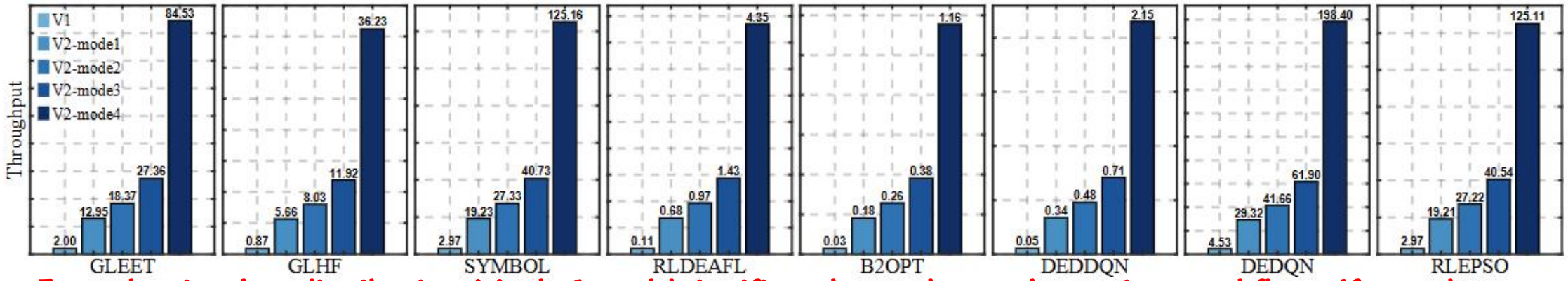
Through the universal interfaces. MetaBox-v2 can surely be compatible and extendable with related benchmarks. This outlines a vision of our work: benchmark zoo system.

For example, MetaBox provides interface for users to extend BBO optimzers from any BBO repo such as PyPop7, which includes hundreds of optimizers. MetaBox also provides interfaces for users to extend optimization problems from exsiting repos such as EvoX, which problem fast implementation of complex/realistic problems.

EXP： parallel acceleration



With a batch_size of 16, the Vectorized Env consistently saves training time for users.
We also found such batched training stabillizes convergence and boost training effectiveness.

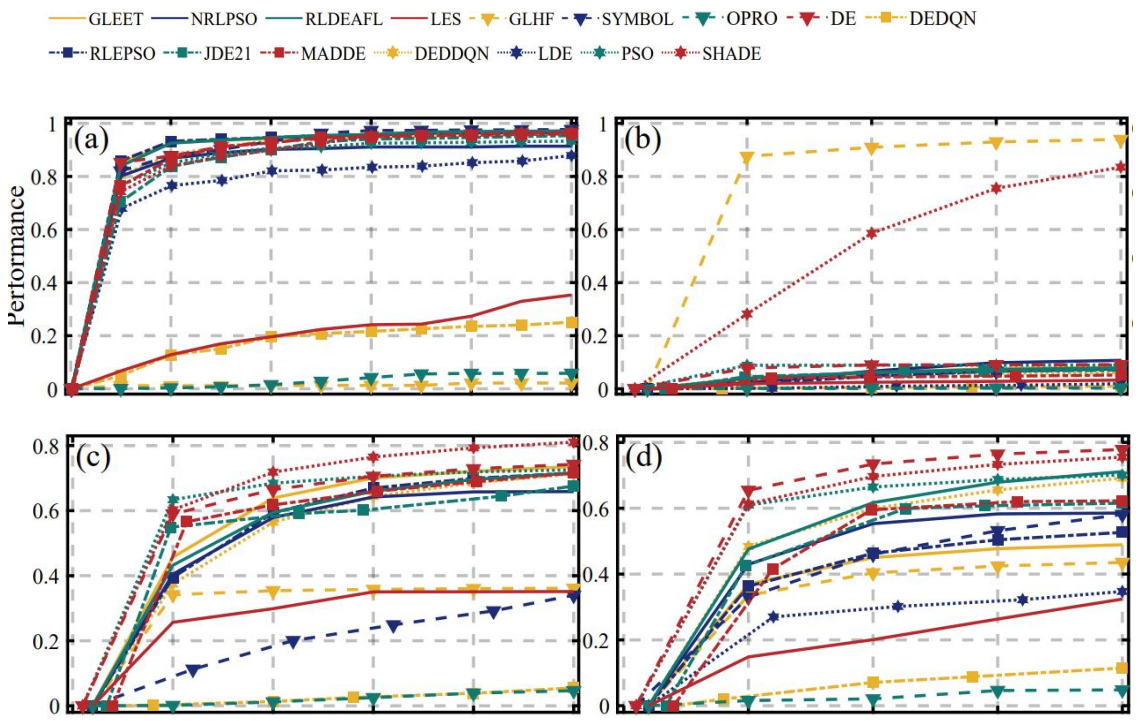


Even the simplest distribution Mode-1 could significantly accelerate the testing workflow. If users have advanced hardware, the distribution Mode-4 could introduce no less than 40x acceleration.

EXP： generalization potential

NEURAL INFORMATION PROCESSING SYSTEMS

	Sharp_Ridge	Different_Powers	Schaffers_HC	Composite_GR	Schwefel	Gallagher_21	Katsuura	Lunacek_BR
PSO(1995) [2]	1.905E+02 ± 2.156E+01	6.802E-01 ± 1.760E-01	5.600E+00 ± 1.368E+00	3.290E+00 ± 5.796E-01	2.560E+00 ± 3.067E-01	6.803E+00 ± 6.472E+00	1.272E+00 ± 2.933E-01	6.139E+01 ± 5.747E+00
DE(1997) [3]	8.588E-01 ± 1.054E+00	8.180E-04 ± 2.537E-04	9.454E-02 ± 6.483E-02	2.577E+00 ± 4.860E-01	9.156E-01 ± 3.039E-01	3.393E+00 ± 4.999E+00	1.467E+00 ± 2.734E-01	4.210E+01 ± 3.043E+00
SHADE(2013) [70]	1.442E+00 ± 4.321E-01	2.721E-04 ± 4.192E-05	2.649E-01 ± 6.818E-02	2.238E+00 ± 3.476E-01	1.338E+00 ± 1.957E-01	1.155E+00 ± 9.320E-01	1.553E+00 ± 3.454E-01	4.248E+01 ± 4.209E+00
JDE21(2021) [71]	3.476E+00 ± 6.350E+00	4.398E-04 ± 3.807E-04	4.496E-01 ± 3.700E-01	2.542E+00 ± 6.355E-01	5.777E-01 ± 2.246E-01	1.604E+00 ± 1.641E+00	1.416E+00 ± 3.359E-01	4.059E+01 ± 7.940E+00
MADDE(2021) [72]	1.736E+00 ± 3.300E-01	5.830E-04 ± 2.318E-04	9.538E-01 ± 2.897E-01	1.077E+00 ± 3.709E-01	8.049E-01 ± 1.997E-01	5.458E-01 ± 7.264E-01	1.350E+00 ± 2.395E-01	4.308E+01 ± 4.974E+00
RNNOPT(2017) [11]	1.822E+03 ±0.000E+00	2.297E+01 ±0.000E+00	4.645E+01 ±0.000E+00	3.609E+00 ±0.000E+00	9.297E+03 ±1.819E-12	8.431E+01 ±0.000E+00	2.186E+00 ±0.000E+00	1.142E+02 ±0.000E+00
DEDDQN(2019) [27]	**1.841E-03 ±1.841E-03**	**4.224E-09 ±4.069E-09**	**1.080E-02 ±7.097E-03**	2.480E+00 ±5.250E-01	1.720E+00 ±4.164E-01	1.574E+00 ±9.236E-01	1.344E+00 ±2.839E-01	4.039E+01 ±4.264E+00
DEDQN(2021) [29]	9.538E+02 ±1.548E+02	1.115E+01 ±2.837E+00	2.709E+01 ±5.790E+00	1.268E+01 ±2.131E+00	4.880E+03 ±3.385E+03	5.711E+01 ±1.366E+01	3.286E+00 ±6.136E-01	1.591E+02 ±2.132E+01
LDE(2021) [28]	5.955E-01 ±5.103E-01	5.159E-05 ±3.700E-05	2.156E-01 ±1.238E-01	2.024E+00 ±1.812E-01	1.071E+00 ±1.603E-01	**4.292E-01 ±7.059E-01**	1.306E+00 ±2.245E-01	3.616E+01 ±3.494E+00
RLPSO(2021) [73]	2.769E+02 ±7.000E+01	1.481E+00 ±9.514E-01	1.429E+01 ±2.968E+00	3.629E+00 ±1.115E+00	2.722E+00 ±2.998E-01	1.597E+01 ±1.719E+01	2.225E+00 ±3.550E-01	6.525E+01 ±7.460E+00
RLEPSO(2022) [30]	6.388E+00 ±6.093E+00	2.554E-04 ±1.396E-04	1.687E+00 ±7.471E-01	1.387E+00 ±4.516E-01	1.261E+00 ±2.497E-01	7.703E+00 ±1.223E+01	1.017E+00 ±2.993E-01	**2.413E+01 ±7.015E+00**
NRLPSO(2023) [52]	1.968E+02 ±8.105E+01	6.449E-01 ±3.607E-01	5.710E+00 ±2.194E+00	3.367E+00 ±1.081E+00	2.631E+00 ±4.837E-01	7.478E+00 ±5.155E+00	1.599E+00 ±4.433E-01	7.007E+01 ±1.466E+01
LES(2023) [15]	1.099E+03 ±1.516E+02	1.273E+01 ±2.222E+00	3.812E+01 ±6.523E+00	1.215E+01 ±2.095E+00	8.044E+03 ±4.741E+03	5.777E+01 ±2.074E+01	4.099E+00 ±9.875E-01	1.793E+02 ±2.481E+01
GLEET(2024) [31]	4.464E+00 ±7.370E+00	1.130E-04 ±8.072E-05	2.137E+00 ±1.618E+00	**8.624E-01 ±3.202E-01**	1.481E+00 ±1.765E-01	8.632E+00 ±1.209E+01	**4.839E-01 ±2.181E-01**	2.717E+01 ±8.473E+00
GLHF(2024) [13]	9.652E+02 ±1.286E+02	1.074E+01 ±1.796E+00	3.163E+01 ±5.541E+00	1.027E+01 ±1.857E+00	6.827E+03 ±4.036E+03	4.923E+01 ±1.678E+01	3.527E+00 ±9.244E-01	1.582E+02 ±2.103E+01
RLDAS(2024) [32]	1.627E+00 ±1.073E+00	3.740E-04 ±2.542E-04	9.798E-01 ±5.450E-01	1.650E+00 ±4.859E-01	**5.505E-01 ±3.074E-01**	4.698E-01 ±7.563E-01	1.296E+00 ±2.623E-01	3.630E+01 ±1.035E+01
SYMBOL(2024) [33]	1.344E+01 ±9.453E+00	5.332E-03 ±2.537E-03	4.256E+00 ±2.238E+00	1.383E+00 ±4.880E-01	1.732E+00 ±2.436E-01	5.611E+00 ±4.981E+00	6.371E-01 ±3.070E-01	3.188E+01 ±1.164E+01
OPRO(2024) [17]	2.003E+03 ± 1.562E+02	3.007E+01 ± 3.326E+00	5.099E+01 ± 9.258E+00	1.434E+01 ± 6.317E+00	9.299E+03 ± 4.804E+03	9.031E+01 ± 2.184E+01	6.113E+00 ± 2.771E+00	1.812E+02 ± 2.562E+01
B2OPT(2025) [12]	2.581E+02 ±3.724E+01	2.510E+00 ±3.641E-01	6.057E+00 ±1.696E+00	8.728E-01 ±2.494E-01	2.543E+00 ±1.547E-01	1.130E+01 ±7.585E+00	1.575E+00 ±2.701E-01	5.814E+01 ±6.917E+00
RLDEAFL(2025) [36]	1.136E+01 ±1.345E+01	1.487E-04 ±9.165E-05	4.166E+00 ±2.214E+00	2.535E+00 ±1.036E+00	1.397E+00 ±3.326E-01	5.452E+00 ±6.106E+00	1.199E+00 ±6.034E-01	3.231E+01 ±7.111E+00
Rank	1:LDE, 2:DEDDQN, 3:RLDAS, 4:SHADE, 5:MADDE, 6:GLEET, 7:RLEPSO, 8:RLDEAFL, 9:JDE21,10:DE, 11:SYMBOL, 12:PSO, 13:B2OPT, 14:NRLPSO, 15:RLPSO, 16:GLHF, 16:DEDQN, 18:RNNOPT, 19:LES, 20:OPRO							



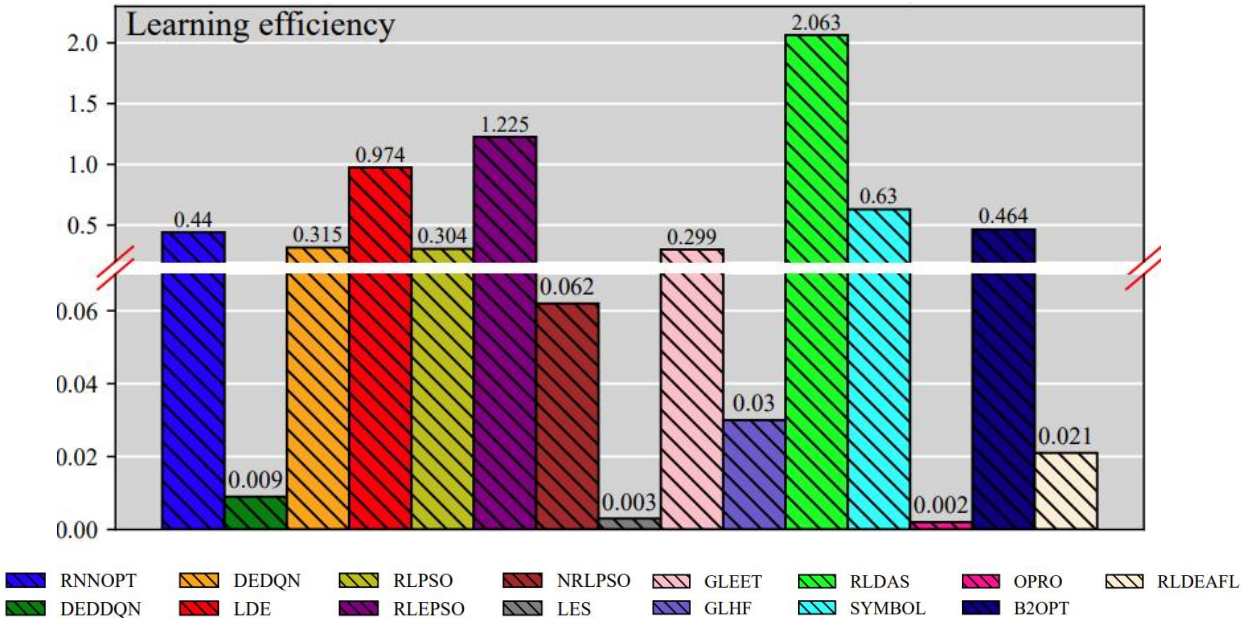
(a) bbob-noisy-30D; (b)protein; (c) uav; and (d) hpob.

Generally, MetaBBO approaches could achieve ideal performance when distribution shift is “TINY” (In-distribution). However, when the distribution shift is “MASSIVE” (out-of-distribution), it is another story..

EXP： ensure evluation objectivity

NEURAL INFORMATION PROCESSING SYSTEMS

Learning Efficiency Indicator. We provide a novel built-in metric in MetaBox-v2: *learning efficiency*, to measure how efficiently a MetaBBO approach learns an effective meta-level policy. Specifically, during the training of a given algorithm $\mathcal{A}$, we save a series of its model snapshots $\{\mathcal{A}^{(g)}\}_{g=0}^{G}$ where G is the number of training epochs. Evaluating all snapshots on the testsuite $\mathbb{D}$, we obtain corresponding metadata: $\{md(\mathcal{A}^{(g)}, \mathbb{D})\}_{g=0}^{G}$. Suppose training $\mathcal{A}^{(g)}$ consumes $T^{(g)}$ hours, then the *learning efficiency* of $\mathcal{A}^{(g)}$ is computed as: $\frac{\text{Perf}(\mathcal{A}^{(g)},\mathbb{D})}{T^{(g)}}$. This metric could fairly reflect the training efficiency of $\mathcal{A}$ at different time slots g.



Most RL & SL approaches present more efficient learning patterns.

Neuroevolution & ICL approaches, however, is far from being efficient pipelines.

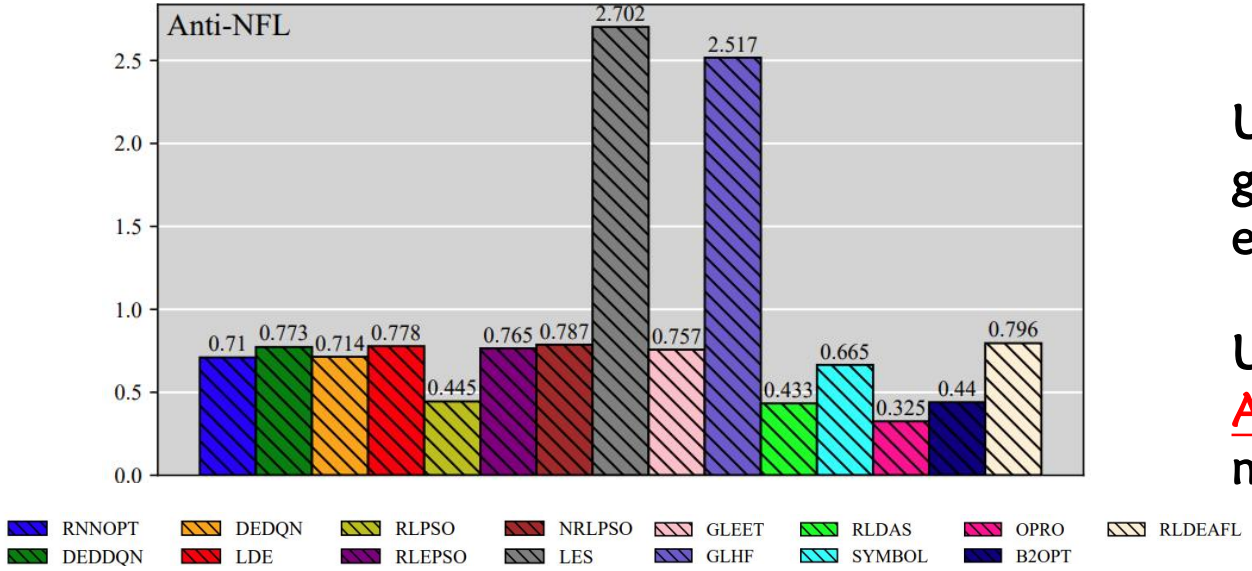
EXP： ensure evluation objectivity

NEURAL INFORMATION PROCESSING SYSTEMS

Anti-NFL Indicator. Recall that the core motivation of MetaBBO is to learn generalizable policy towards unseen problems. This is somewhat against the well-known *no-free-lunch* theorem [5]. We hence propose a novel indicator named as Anti-NFL, which could reflect the performance variance of a given algorithm $\mathcal{A}$ on unseen testsuites apart from the one it was trained. Suppose $\mathcal{A}$ is trained on $\mathbb{D}_{\text{train}}$, and tested on other B testsuites: $\{\mathbb{D}_{\text{test}}^{(b)}\}_{b=1}^{B}$. After obtaining all of the metadata by testing the final model $\mathcal{A}^{(G)}$ on these testsuites, the Anti-NFL is computed as:

$$\text{Anti-NFL} = \exp\left(\frac{1}{B}\sum_{b=1}^{B}\frac{\text{Perf}\left(\mathcal{A}^{(G)}, \mathbb{D}_{\text{test}}^{(b)}\right) - \text{Perf}\left(\mathcal{A}^{(G)}, \mathbb{D}_{\text{train}}\right)}{\text{Perf}\left(\mathcal{A}^{(G)}, \mathbb{D}_{\text{train}}\right)}\right) \tag{2}$$

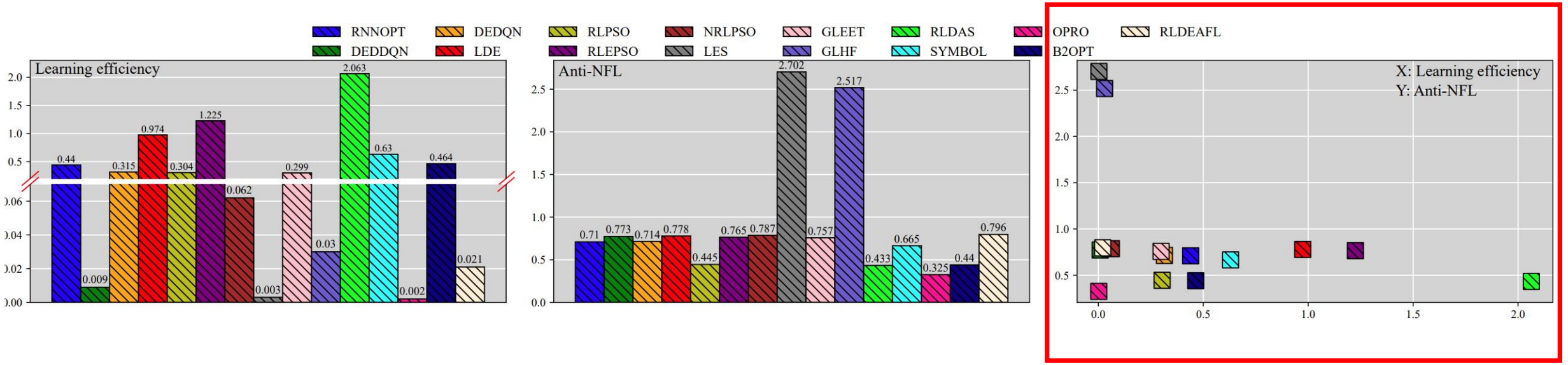
A larger Anti-NFL indicator indicates that $\mathcal{A}$ performs robustly under problem-shifts, and vice versa.



Under this interesting metric, we found that although Neuroevolution method LES is inefficient, however, its Anti-NFL is high.

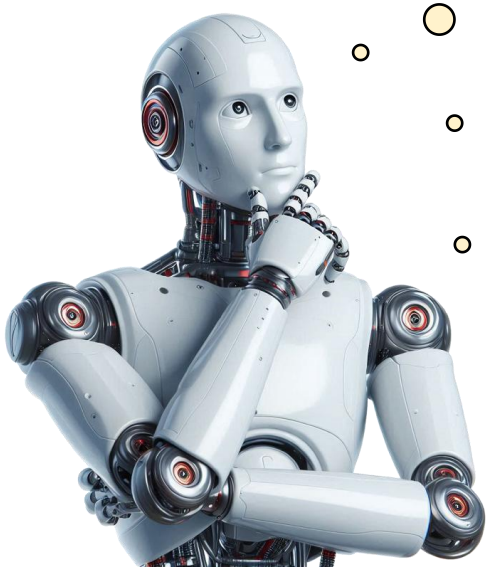
Unfortunately, an MetaBBO approach with high Anti-NFL does not indicate its superiority, since it may performs bad on both train and test sets.

EXP： ensure evluation objectivity



Combining the results of learning efficiency and Anti-NFL indicator, we analyze the domination relationship of MetaBBO baselines. It can be observed that, so far, no baseline participating in this case study dominates all the others. This reflects that there is certain design tradeoff in existing MetaBBO between the efficiency and effectiveness.

Challenges & Oppotunity:



Distribution Shift: Where, Why & How

Model Capacity & Data Scale

Effectiveness & Efficiency Tradeoff

TAKEAWAYS

1. EXISTING LITERATURE ONLY PROVIDES NARROW EVALUATION, NOT FRIENDLY FOR NEWCOMERS.

2. OVERFITTING OR NOT IS STILL A MYSTERY IN MANY METABBO WORK.

3. WE SHOULD NEVER GIVE UP ON DISCUSSING HOW TO MEASURE CAPABILITY OF LEARNING SYSTEM, E.G., METABBO SYSTEMS.

Thanks

NEURAL INFORMATION PROCESSING SYSTEMS

METAEVO

MetaEvolution (MetaEvo) Group is funded by Prof. Yue-Jiao Gong, Associate Editor of IEEE Transaactions on Evolutionary Computation. Our aim is to explore the possibility of learning-assisted, meta-learned optimization techniques that show general purpose instead human-crafted specialization. We started from 2021 and now this topic has been a hot-press in optimization community.

Team Page

Project Page

My Scholor Page

My favorite sentence: “**Information is what generates information, say, evolution.**”

My favorite sentence heard from others: “**Things won are done; joy’s soul lies in the doing.**”