

Mix & match your *model*, *algorithm*, and *task* and start to BenchMARL!

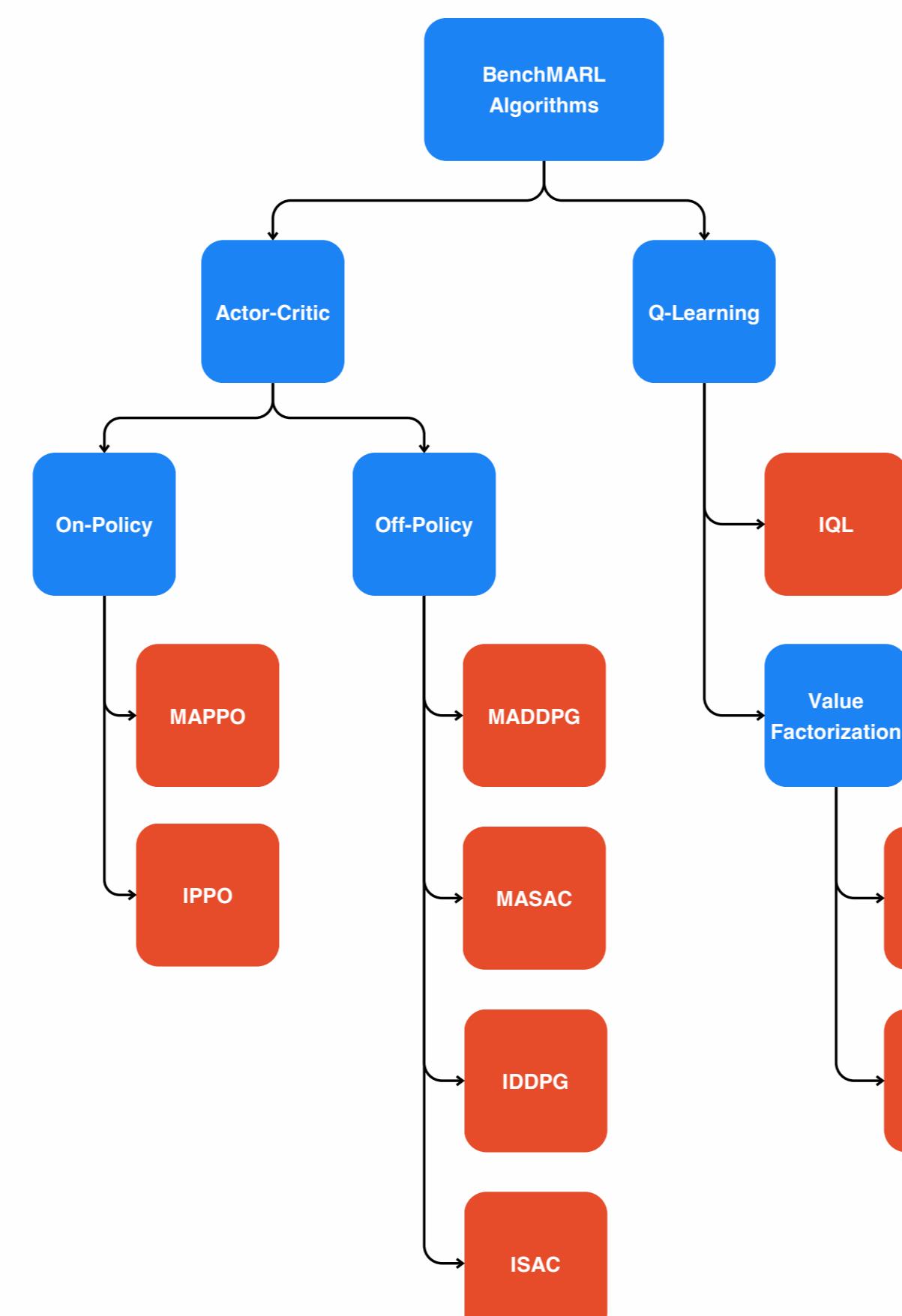
### Models

Models are neural network blueprints that can be chained to make actors and critics with or without parameter sharing.

Model	Policy	Critic	Local input	Global input
MLP	✓	✓	✓	✓
CNN	✓	✓	✓	✓
GNN	✓	✓	✗	
Deep Sets	✓	✓	✓	✓

### Algorithms

Algorithms are an ensemble of components (e.g., loss, replay buffer, exploration strategy) that determine the training strategy.

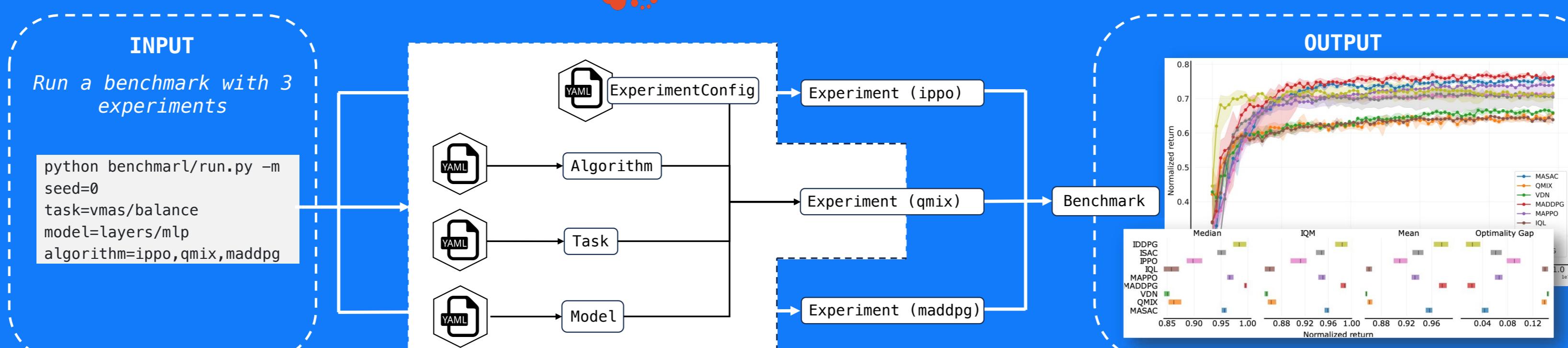


### Tasks

Tasks are scenarios from a simulated environment which constitute the MARL challenge to solve.

Environment	# Tasks
VMAS	27
SISL	3
SMACv2	15
MPE	8
Melting Pot	49

## BenchMARL



### Reporting and plotting

Automatically report and plot your BenchMARL results using statistically-strong integrated tools [1,2].

Interactively log your results on Wandb and compare with the publicly available benchmarks:



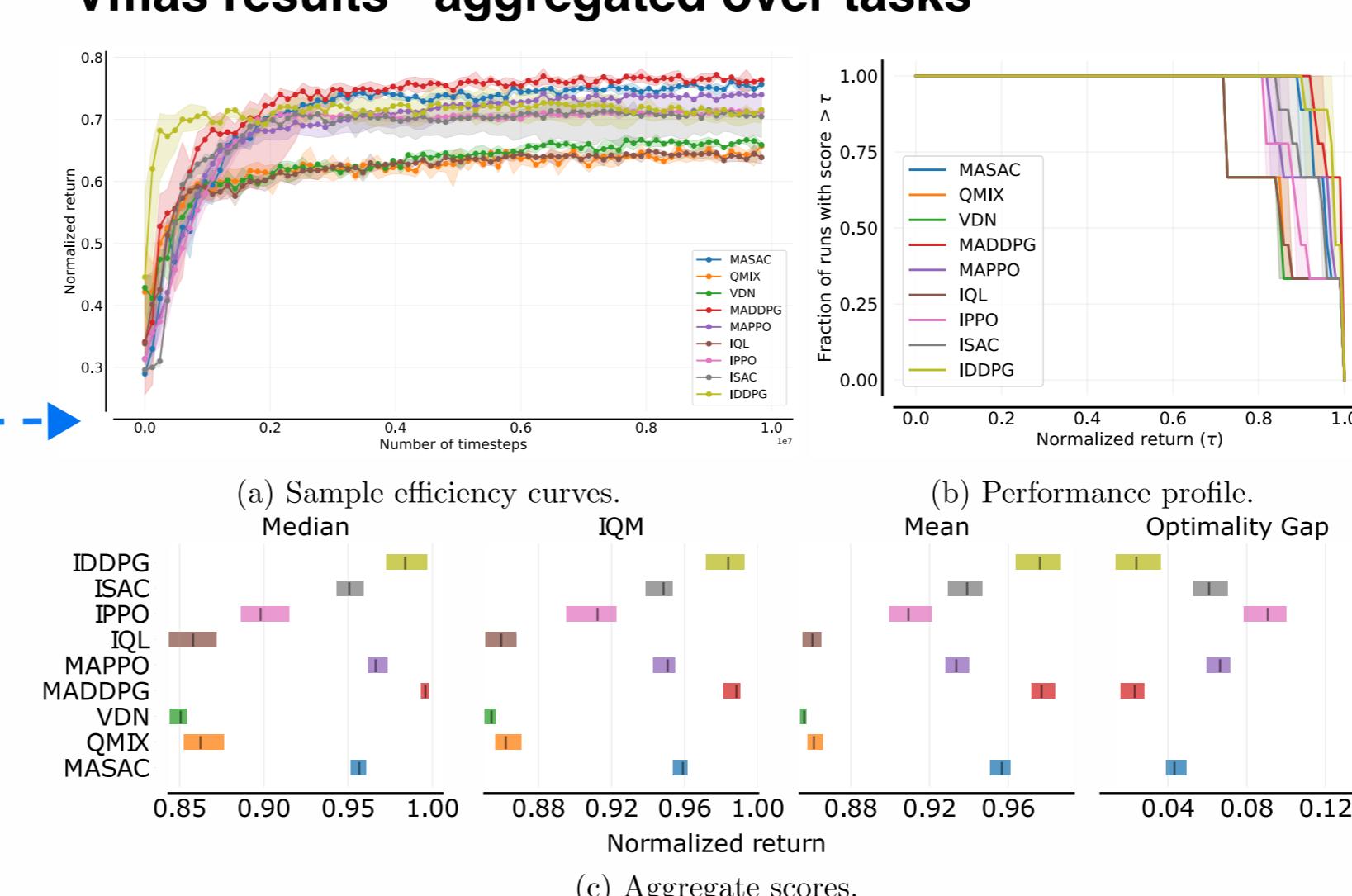
We are benchmarking and releasing results for BenchMARL tasks.

Check out the results in the VMAS environment:

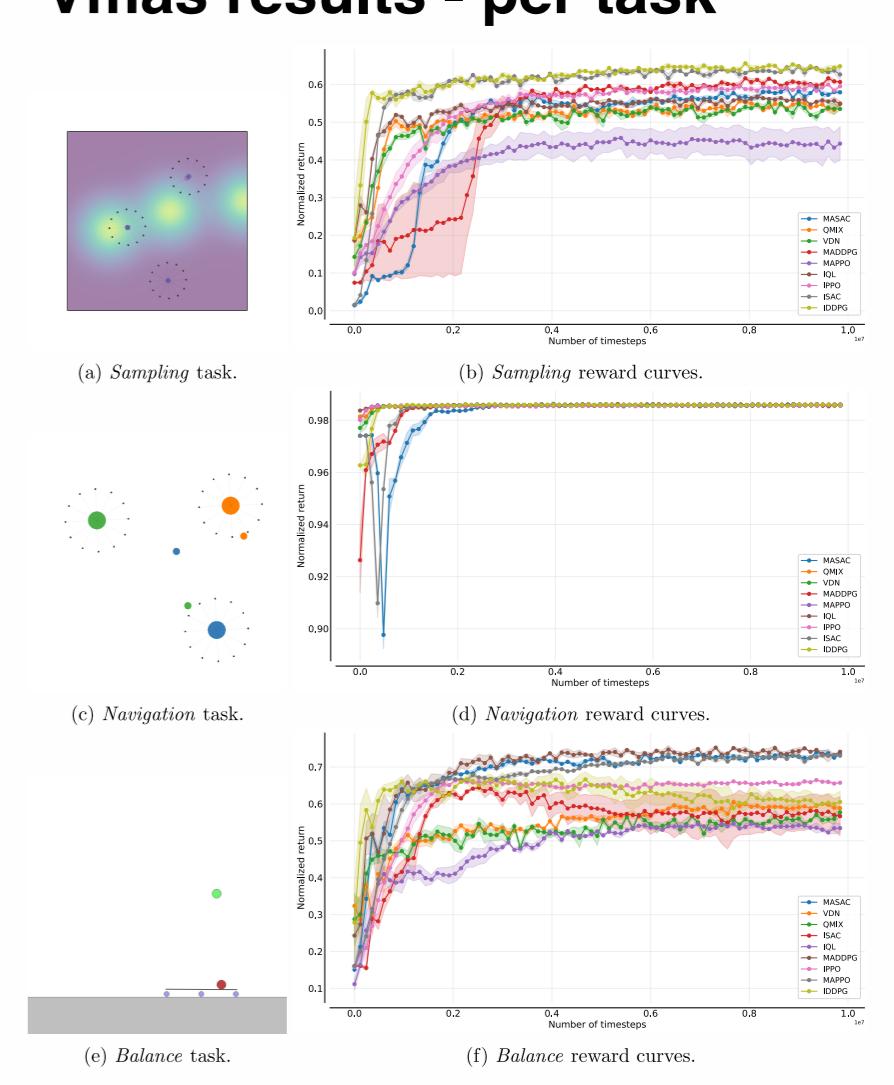
[1] Agarwal, Rishabh, et al. "Deep reinforcement learning at the edge of the statistical precipice." Advances in neural information processing systems 34 (2021): 29304-29320.

[2] Gorse, Rihab, et al. "Towards a standardised performance evaluation protocol for cooperative marl." Advances in Neural Information Processing Systems 35 (2022): 5510-5521.

### Vmas results - aggregated over tasks



### Vmas results - per task



### Configuring

BenchMARL uses **Hydra** to load configurations from YAML files into Python dataclasses. This allows to easily override and sweep parameters while decoupling them from the codebase.

### Extending

Each component in the library has an associated abstract class. This makes it easy to integrate **custom algorithms**, **models**, and **tasks**, allowing to compare them against the wide repository of already implemented ones.

### Callbacks and checkpointing

We support custom callbacks in various phases of training. Experiments can also be checkpointed and reloaded for evaluation or deployment on robotic platforms.

### Documentation and tests

Component documentation is available online and full training integration tests are run for each task-algorithm combination.

### masac.yaml

```
defaults:
  - masac_config
  - self_
share_param_critic: True
num_value_nets: 2
loss_function: "l2"
delay_value: True
target_entropy: "auto"
discrete_target_entropy_weight: 0.2
alpha_init: 1.0
min_alpha: null
max_alpha: null
fixed_alpha: False
scale_mapping: "biased_softplus_1.0"
```

YAML configuration file for the MASAC algorithm

### masac.py

```
@dataclass
class MasacConfig(AlgorithmConfig):
    share_param_critic: bool = MISSING
    num_value_nets: int = MISSING
    loss_function: str = MISSING
    delay_value: bool = MISSING
    target_entropy: Union[float, str] = MISSING
    discrete_target_entropy_weight: float = MISSING
    alpha_init: float = MISSING
    min_alpha: Optional[float] = MISSING
    max_alpha: Optional[float] = MISSING
    fixed_alpha: bool = MISSING
    scale_mapping: str = MISSING
```

Python dataclass for the MASAC algorithm

### Features

BenchMARL's features focus on enabling its core tenets: standardization and reproducibility.

BenchMARL core design guidelines:

- Reproducibility
- Standardized reporting
- TorchRL backend
- Experiment independence
- Easy integration of new solutions

The TorchRL backend allows BenchMARL to re-use extensively-benchmarked single-agent implementations.