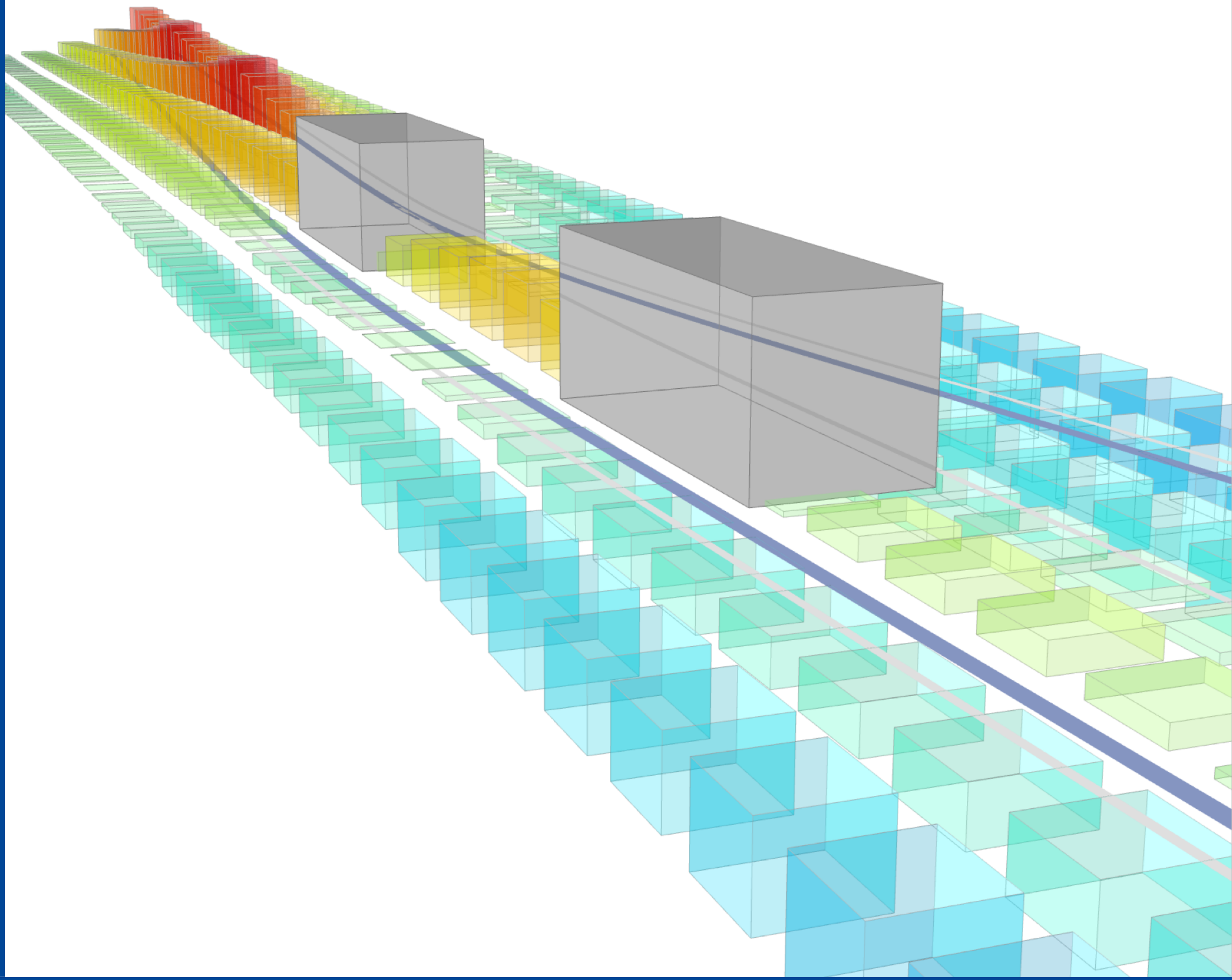
BARK – Developing, Simulating and Benchmarking Behavior Planners

Patrick Hart

fortiss GmbH

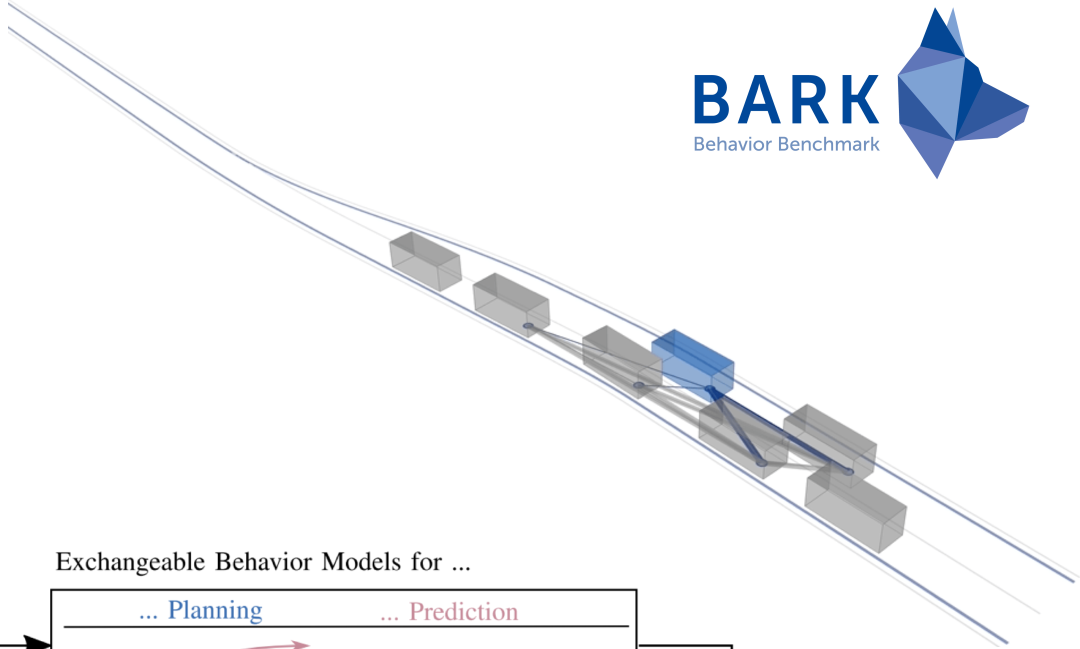


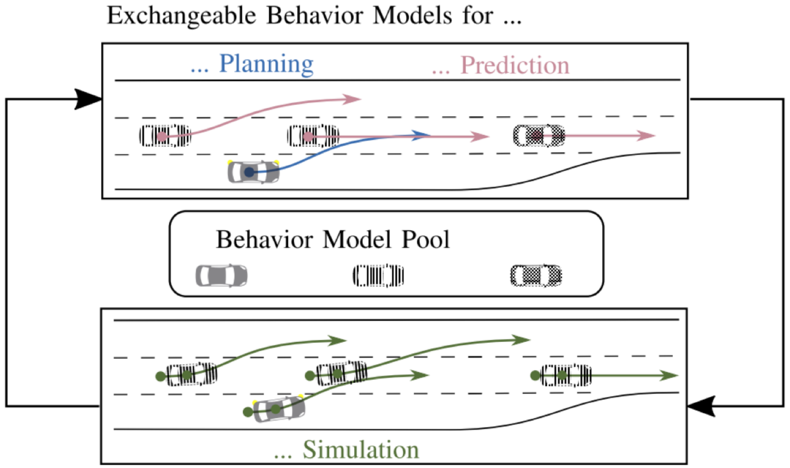
BARK: Introduction

Motivation: Benchmarking

- BARK is a framework for developing and continuously benchmarking behavior planners
- BARK closes the gap of systematic and continuous benchmarking
- Core characteristics:
  - Interaction-aware, fast, semantic simulation
  - Models are used for simulation, prediction, and planning

BARK
Behavior Benchmark





BARK: Introduction

Motivation

- Aimed at three persona:
  - **Python Evangelist**: Implementing Python behavior models
  - **ML Aficionado**: Uses BARK-ML for learning behaviors
  - **C++ Enthusiast**: Developing C++ behavior models
- Benchmark who built the best **BehaviorModel!**

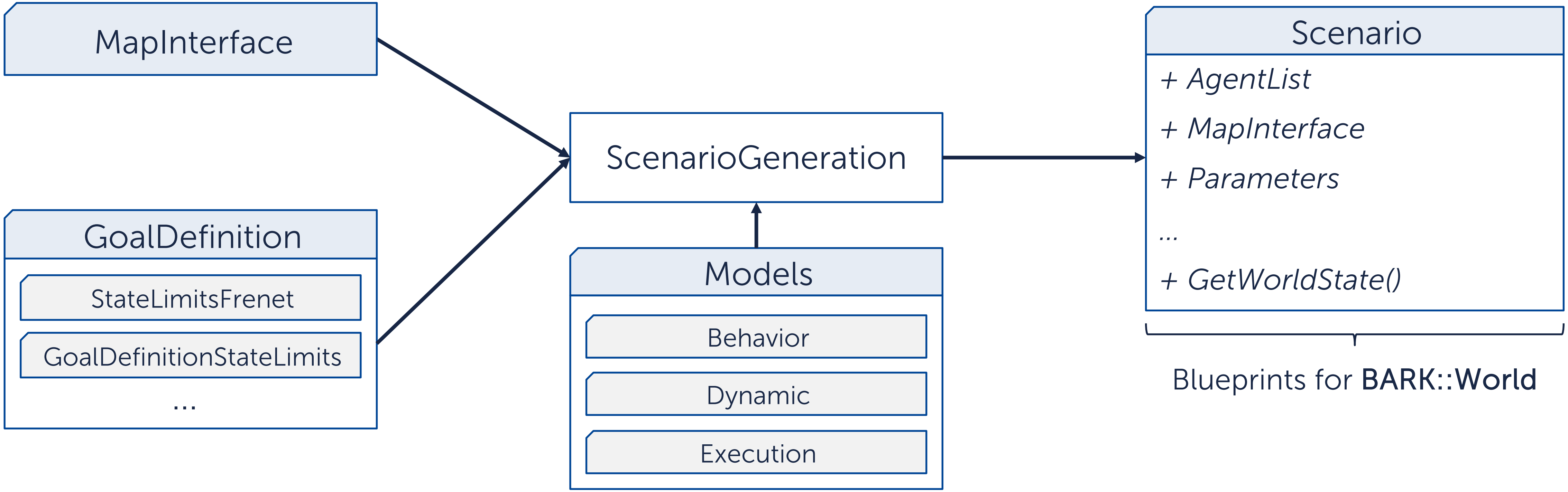
python™

TensorFlow

PyTorch

BARK: Introduction

Overview



Overview

1. Python Evangelist

   Implementation of a Python BehaviorModel

2. Machine Learning Aficionado

   OpenAI-Gym environments; BARK-ML Agents; …

3. C++ Enthusiast

   Building from source; Efficient C++ Behavior Model

4. Benchmarking

   Benchmarking Behaviors in BARK

5. Summary

Python Evangelist

Creating Behavior Models

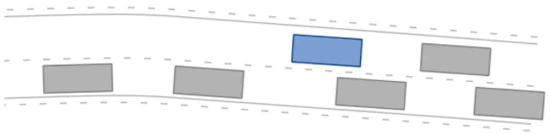
Open in Colab

pip install bark-simulator

```python
class DerivedBehaviorModel(BehaviorModel):
  def __init__(self,
               params=ParameterServer(),
               plan_fn=None):
    BehaviorModel.__init__(self, params)
    self.plan_fn = plan_fn

  def Plan(
      self,
      step_time: np.array,
      observed_world: ObservedWorld
  ) -> np.ndarray:
    """Plans a trajectory for an agent based on the ObservedWorld."""
    trajectory = plan_fn(observed_world, self.params)

    # store trajectory for, e.g., replanning
    super().SetLastTrajectory(trajectory)
    return trajectory

  def Clone(
      self
  ) -> BehaviorModel:
    """Clone function specifying how the model shall be cloned."""
    return self
```

```python
def plan_fn(
    observed_world: ObservedWorld,
    params: ParameterServer
) -> np.ndarray:
  """Function that returns trajectory based on the ObservedWorld."""
  time, x, y, theta, v = list(observed_world.ego_state)

  # return trajectory
  return np.array([[time + 0.,   x, y, theta, v],
                   [time + 0.2,  x, y, theta, v],
                   [time + 0.4,  x, y, theta, v]])
```

```python
# run behavior model
derived_behavior_model = DerivedBehaviorModel(plan_fn=plan_fn)
world = get_bark_world(derived_behavior_model)
for _ in range(0, 5):
  world.Step(0.2)
```



Python Evangelist

Creating Behavior Models

Open in Colab

```python
class DerivedBehaviorModel(BehaviorModel):
  def __init__(self,
               params=ParameterServer(),
               plan_fn=None):
    BehaviorModel.__init__(self, params)
    self.plan_fn = plan_fn

  def Plan(
      self,
      step_time: np.array,
      observed_world: ObservedWorld
  ) -> np.ndarray:
    """Plans a trajectory for an agent based on the ObservedWorld."""
    trajectory = plan_fn(observed_world, self.params)

    # store trajectory for, e.g., replanning
    super().SetLastTrajectory(trajectory)
    return trajectory

  def Clone(
      self
  ) -> BehaviorModel:
    """Clone function specifying how the model shall be cloned."""
    return self
```

```python
def plan_fn_with_dynamic_model(
    observed_world: ObservedWorld,
    params: ParameterServer
) -> np.ndarray:
  """Function that returns trajectory based on the ObservedWorld."""
  single_track_model = SingleTrackModel()
  state = observed_world.ego_state
  action = np.array([0., 0.])

  # generate trajectory with constant acceleration and steering-rate
  trajectory = []
  for _ in range(0, 5):
    x_dot = single_track_model.stateSpaceModel(state, action)
    state = state + dt*x_dot
    trajectory.append(state)

  # return trajectory
  return np.array(trajectory)
```

- Various of utility functions (e.g., geometry, Frenet, routing, etc.)

Overview

1. Python Evangelist

   Implementation of a Python BehaviorModel

2. Machine Learning Aficionado

   OpenAI-Gym environments; BARK-ML Agents

   BARK Machine Learning

3. C++ Enthusiast

   Building from source; Efficient C++ Behavior Model

4. Benchmarking

   Benchmarking Behavior Models

5. Summary

ML Aficionado

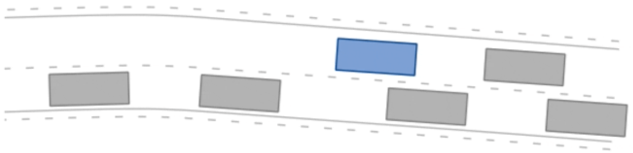
BARK Machine Learning

Open in Colab

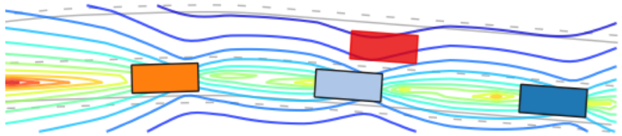
pip install bark-ml

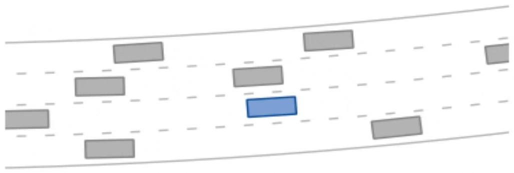
- OpenAI-Gym environments

```python
# import required packages
import gym
import numpy as np
# registers bark-ml environments
import bark_ml.environments.gym
import matplotlib.pyplot as plt

# generate the gym environment
env = gym.make("merging-v0")

# step until terminal
observed_state = env.reset()
terminal = False
while terminal is False:
  action = np.array([0., 0.]) # steering-rate and acceleration
  observed_state, reward, terminal, info = env.step(action)
  print(f"Observed state: {observed_state}, action: {action}, "
        f"reward: {reward}, terminal: {terminal}")
```

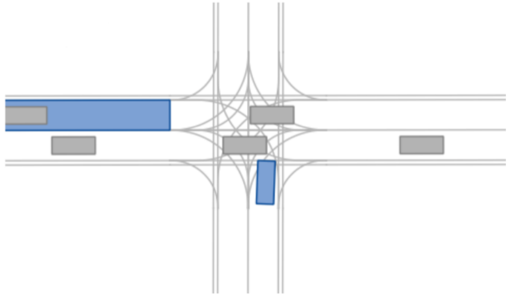


merging-v0





highway-v0

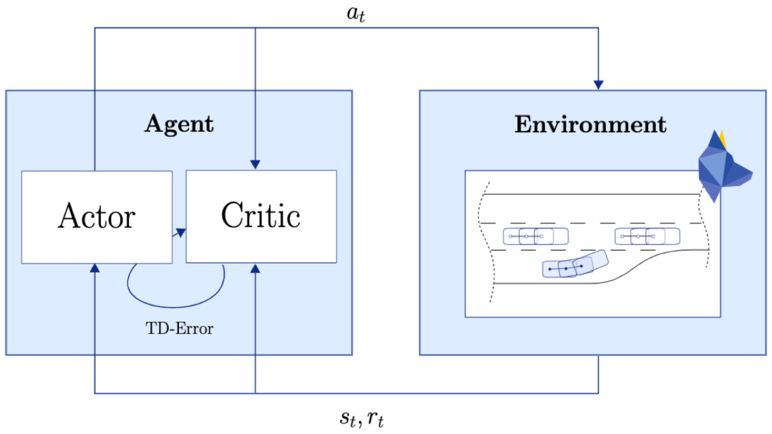


Intersection-v0

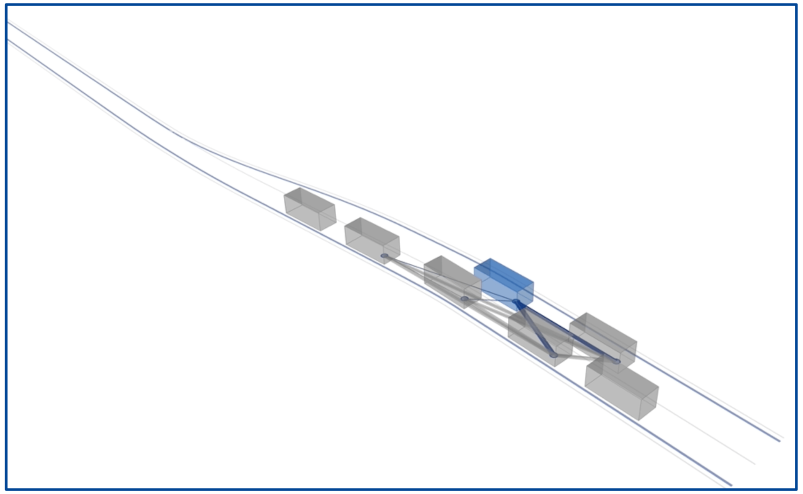
ML Aficionado

BARK-ML Agent Models

- Actor-Critic Agents:
  - Proximal Policy Optimization (PPO)
  - Soft-Actor-Critic (SAC)
  - Graph Neural Network-SAC (GNN-SAC)[1]
  - …



- Quantile Agents:
  - Fully Parameterized Quantile Function (FQF)
  - Implicit Quantile Networks (IQN)
  - …



[1]Hart, Patrick, and Alois Knoll. "Graph Neural Networks and Reinforcement Learning for Behavior Generation in Semantic Environments." *2020 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2020.

ML Aficionado

Training an Agent

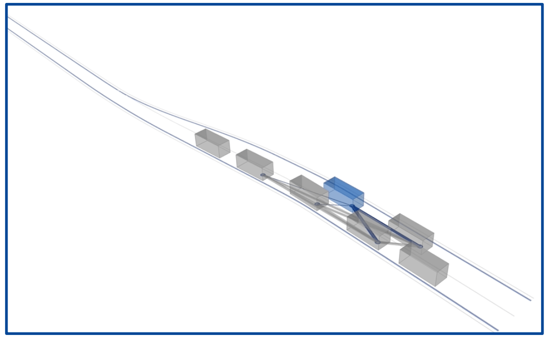
Open in Colab

```
params = ParameterServer()
sac_agent = BehaviorSACAgent(
  environment=env,
  params=params)

env.ml_behavior = sac_agent

runner = SACRunner(
  params=params,
  environment=env,
  agent=sac_agent)

# train
runner.Train()

# visualize results
runner.Run(num_episodes=5, render=True)
```

Create an SAC Agent

Assign agent to the environment

Runner for training, evaluation, and visualization



Overview

1. Python Evangelist

   Implementation of a Python BehaviorModel

2. Machine Learning Aficionado

   OpenAI-Gym environments; BARK-ML Agents; …

3. C++ Enthusiast

   Building BARK from source; Efficient C++ Behavior Model

4. Benchmarking

   Benchmarking Behaviors in BARK

5. Summary

C++ Enthusiast

BARK Core

- Actual BARK World core in C++

```cpp
class World : public commons::BaseType {
 public:
  explicit World(const commons::ParamsPtr& params);
  explicit World(const std::shared_ptr<World>& world);
  virtual ~World() {}

  std::vector<ObservedWorld> Observe(
    const std::vector<AgentId>& agent_ids) const;
  void PlanAgents(const double& delta_time);
  void Execute(const double& delta_time);
  void Step(const float& delta_time);
  virtual std::shared_ptr<World> Clone() const;
  ...
 private:
  world::map::MapInterfacePtr map_;
  AgentMap agents_;
  ObjectMap objects_;
  std::map<std::string, EvaluatorPtr> evaluators_;
  ObserverModelPtr observer_;
  double world_time_;
  ...
};
```

- Behavior models with high computational complexity can utilize C++
- Various C++ **BehaviorModels** available:
  - Lane-following (IDM, MOBIL)
  - Search-based (MCTS)
  - ...
- Using bazel.build, C++ is automatically compiled

```
bazel build //bark/world:world
```

C++ Enthusiast

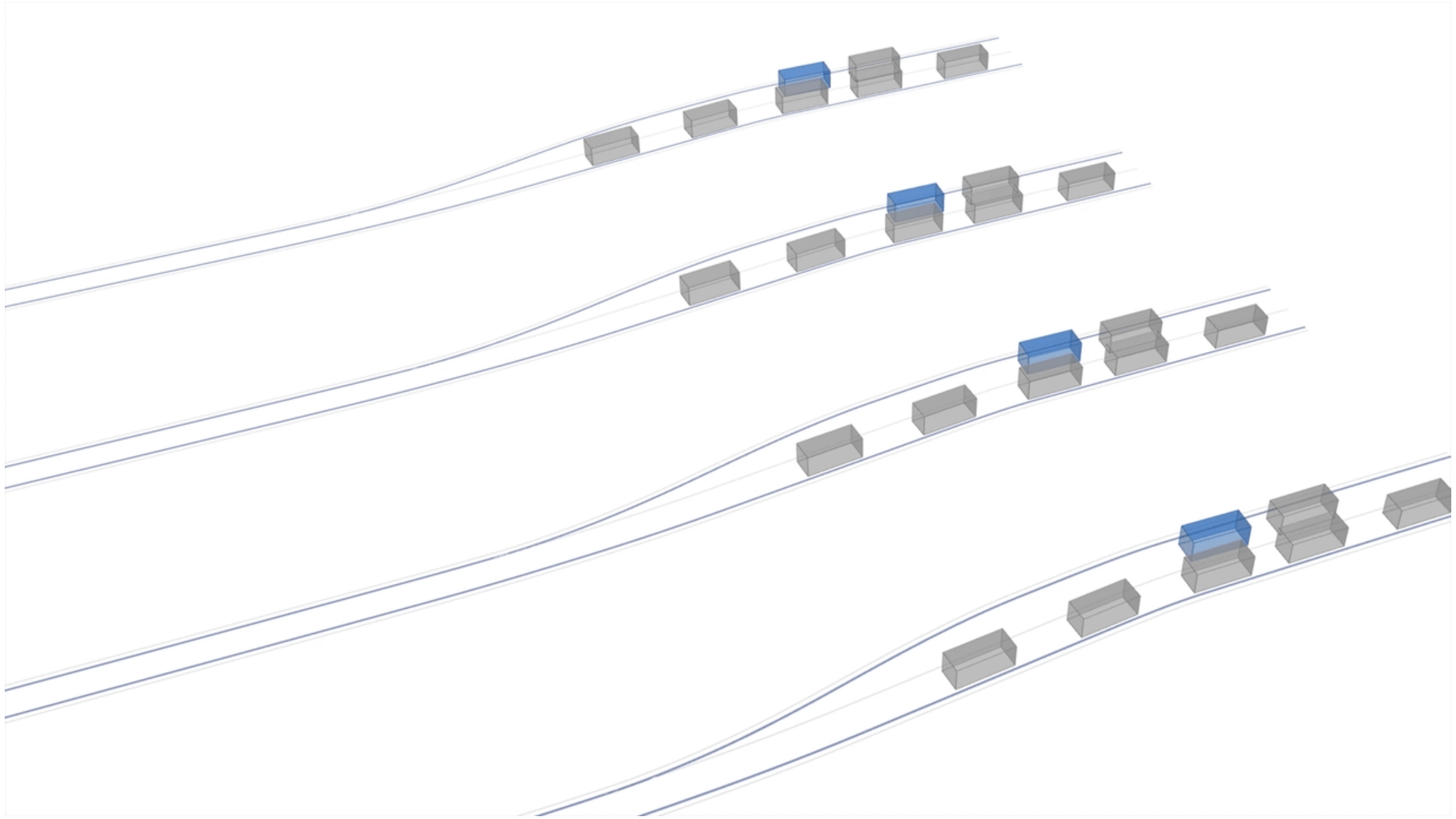
Behavior Model

```cpp
class BehaviorModel : public bark::commons::BaseType {
 public:
  explicit BehaviorModel(
    const commons::ParamsPtr& params,
    BehaviorStatus status) :
      commons::BaseType(params),
      last_trajectory_(),
      last_action_(),
      behavior_status_(status),
      measure_solution_time_(false),
      last_solution_time_(0.0) {}

  virtual Trajectory Plan(
    double min_planning_time,
    const world::ObservedWorld& observed_world) = 0;

  virtual std::shared_ptr<BehaviorModel> Clone() const = 0;
  ...
};
```

ParameterServer is completely serializable

```cpp
Trajectory BehaviorModel::Plan(
  double min_planning_time,
  const world::ObservedWorld& observed_world) {
  auto ego_s = observed_world.CurrentEgoState();
  double dt = 0.2;
  Trajectory trajectory(4, 5);
  for (int = 0; i < trajectory.rows(); i++) {
    trajectory.row(i) = ego_s;
    trajectory[i, 0] = ego_s[0] + i*dt;
    trajectory[i, 1] = ego_s[1] + i*2;
  }
  return trajectory;
}
```

Who built the best behavior model?



- BARK offers systematic behavior benchmarking capabilities

Overview

1. Python Evangelist

   Implementation of a Python BehaviorModel

2. Machine Learning Aficionado

   OpenAI-Gym environments; BARK-ML Agents; ...

3. C++ Enthusiast

   Building from source; Efficient C++ Behavior Model

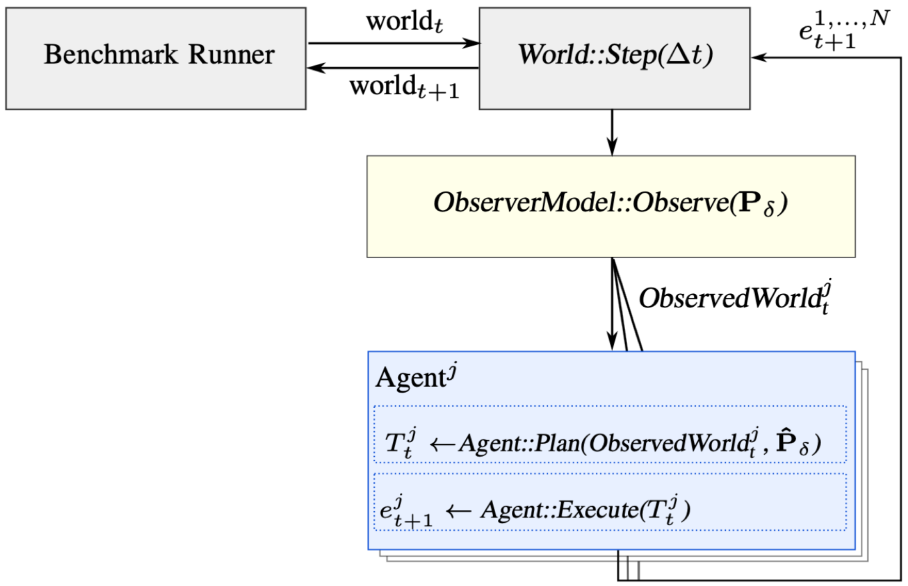
4. Benchmarking

   Benchmarking Behaviors in BARK

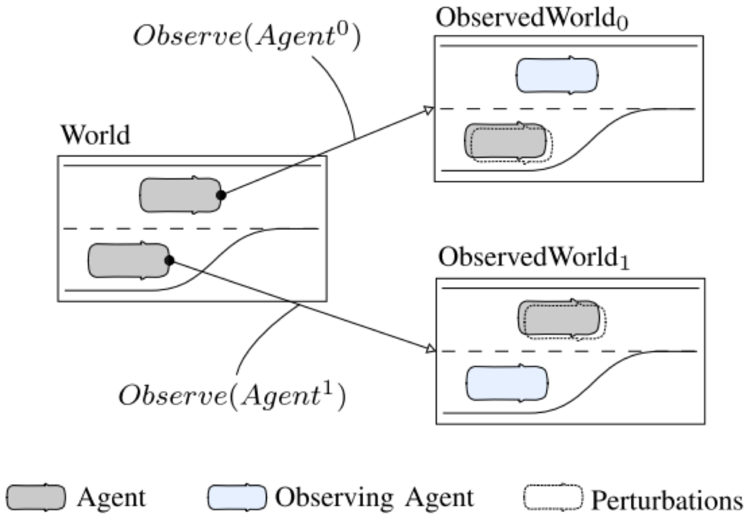
5. Summary

Benchmarking
Concept

- Each agent receives an *ObservedWorld* to plan in



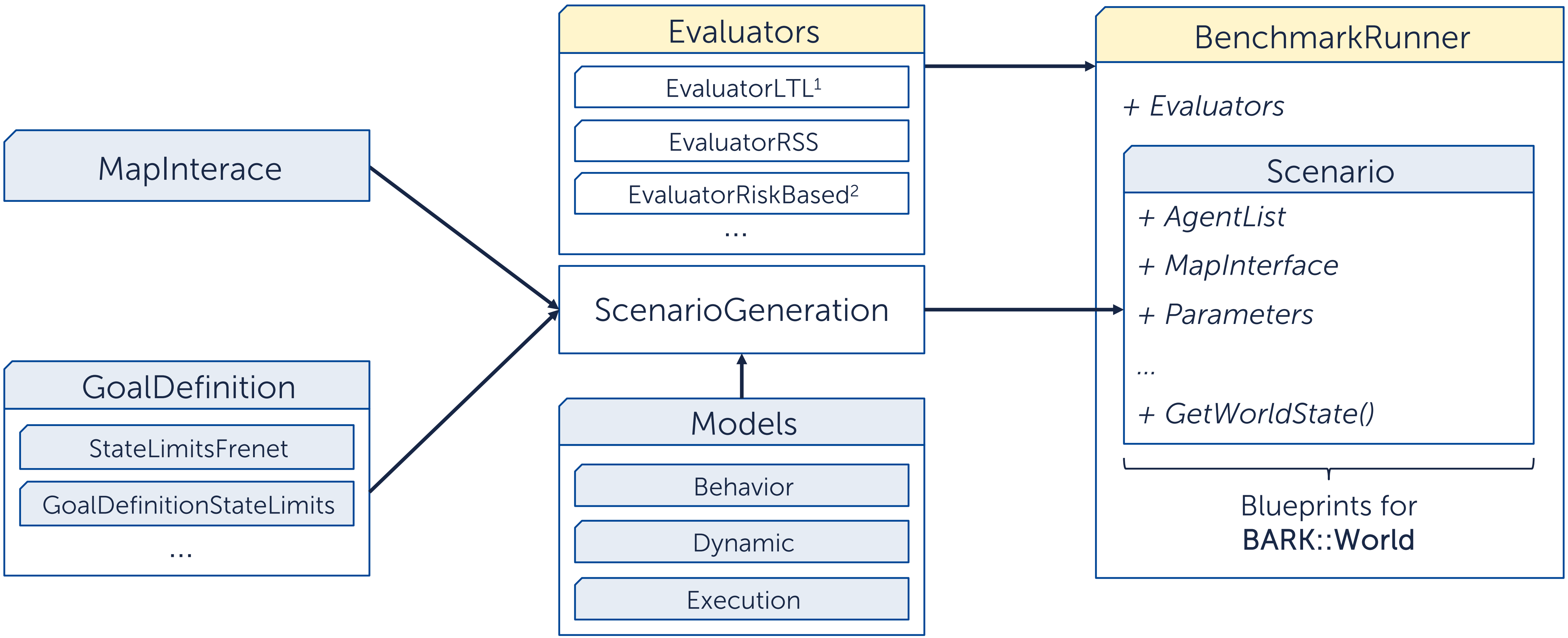
- *ObservedWorld* used to model perturbations, such as uncertainties



J. Bernhard and A. Knoll, "Robust Stochastic Bayesian Games for Behavior Space Coverage," presented at the Robotics: Science and Systems (RSS), Workshop on Interaction and Decision-Making in Autonomous-Driving, 2020.

Benchmarking

Overview: Recap and Extension

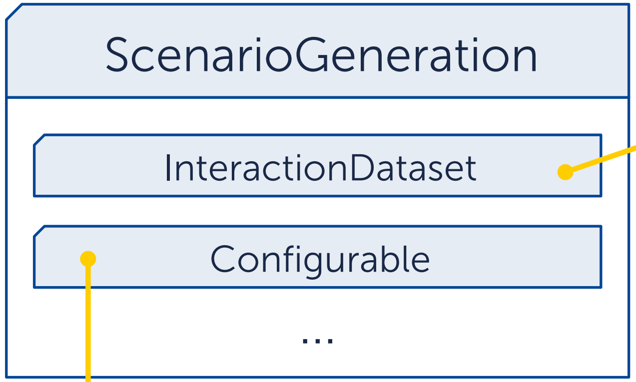


[1] EvaluatorLTL: Esterle, Klemens, Luis Gressenbuch, and Alois Knoll. "Formalizing traffic rules for machine interpretability." CAVS2020.

[2] EvaluatorRiskBased: Bernhard, Julian, and Alois Knoll. "Risk-Constrained Interactive Safety Under Behavior Uncertainty for Autonomous Driving", IV2021.

Benchmarking

ScenarioGeneration

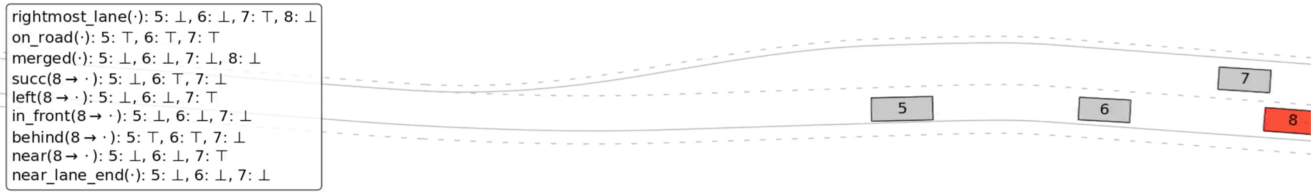


Sampling-based scenario generation with conflict resolution

- INTERACTION dataset integration

DR_DEU_Merging_MT

DR_CHN_Merging_ZS



Esterle, Klemens, Luis Gressenbuch, and Alois Knoll. "Formalizing traffic rules for machine interpretability." *2020 IEEE 3rd Connected and Automated Vehicles Symposium (CAVS)*. IEEE, 2020.

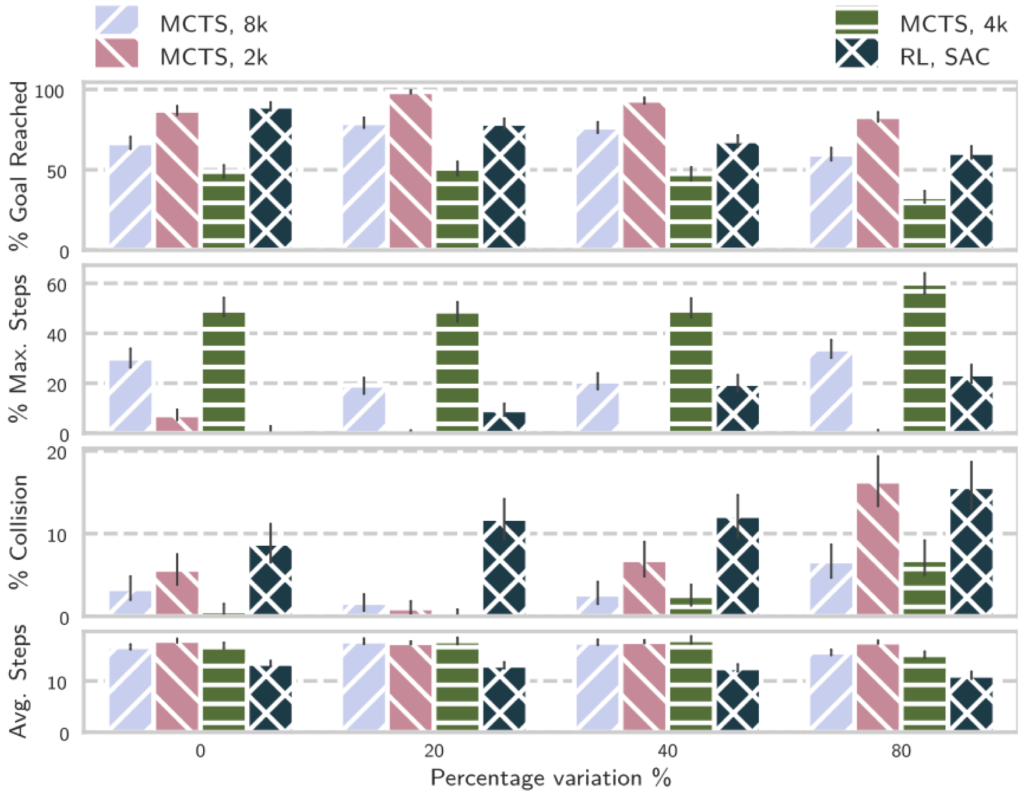
Benchmarking

Example

```
db = BenchmarkDatabase(
  database_root="external/benchmark_database_release")

evaluators = {
  "success" : EvaluatorGoalReached,
  "collision" : EvaluatorCollisionEgoAgent,
  "max_steps": EvaluatorStepCount
}

terminal_when = {
  "collision": lambda x: x, "max_steps": lambda x : x>70
}

behaviors_to_evaluate = {
  "eval_simple": DerivedBehaviorModel(plan_fn=plan_fn),
  "eval_dynamic_model": DerivedBehaviorModel(
    plan_fn=plan_fn_with_dynamic_model)
}

benchmark_runner = BenchmarkRunner(
  benchmark_database=db,
  evaluators=evaluators,
  terminal_when=terminal_when,
  behaviors=behaviors_to_evaluate)

benchmark_runner.run(1000)
benchmark_runner.dataframe.to_pickle("results.pickle")
```

BARK
Benchmark Database

https://github.com/bark-simulator/example_benchmark



Bernhard et al. "BARK: Open behavior benchmarking in multi-agent environments." *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020.

Overview

1. Python Evangelist

   Implementation of a Python BehaviorModel

2. Machine Learning Aficionado

   OpenAI-Gym environments; BARK-ML Agents; …

3. C++ Enthusiast

   Building from source; Efficient C++ Behavior Model

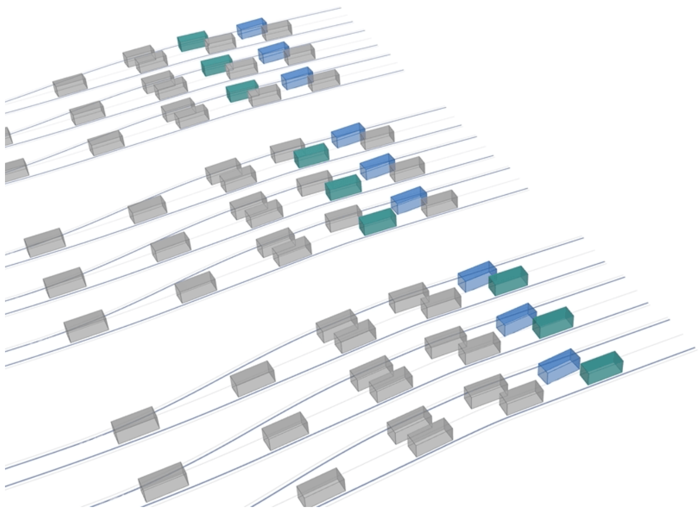
4. Benchmarking

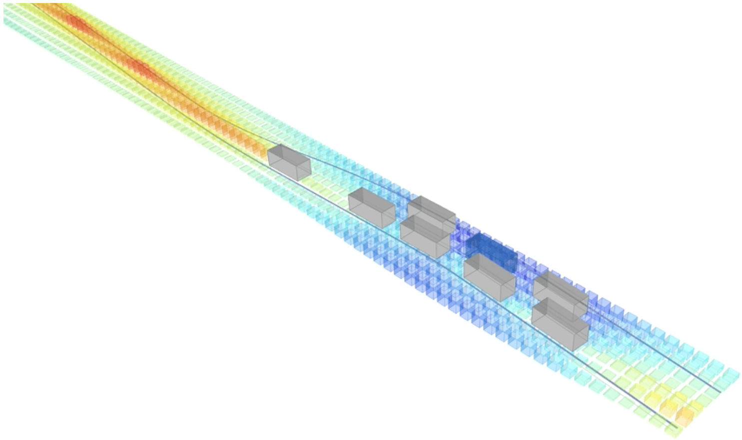
   Benchmarking Behaviors in BARK

5. Summary

Summary

- (rapid) Development of interaction-aware behavior planning algorithms
- Systematic and reproducible benchmarking capabilities
- State-of-the-art research for autonomous driving
- Open-source framework under the MIT license





BARK Behavior Benchmark

https://github.com/bark-simulator/bark

https://github.com/bark-simulator/bark-ml

M.Sc. Patrick Hart

Contact

fortiss GmbH
Guerickestraße 25
80805 München

Patrick Hart
hart@fortiss.org